
# NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**PERFORMANCE ANALYSIS OF WIRELESS NETWORKS FOR INDUSTRIAL AUTOMATION-PROCESS AUTOMATION (WIA-PA)**

by

Brandon Wyatt

September 2017

| | |
|---|---|
| Thesis Advisor: | Preetha Thulasiraman |
| Second Reader: | John McEachen |

THIS PAGE INTENTIONALLY LEFT BLANK

# REPORT DOCUMENTATION PAGE

*Form Approved OMB No. 0704–0188*



| 1. AGENCY USE ONLY *(Leave Blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | September 2017 | Master's Thesis 06-21-2015 to 09-22-2017 |

**4. TITLE AND SUBTITLE**

PERFORMANCE ANALYSIS OF WIRELESS NETWORKS FOR INDUSTRIAL AUTOMATION-PROCESS AUTOMATION (WIA-PA)

**5. FUNDING NUMBERS**

**6. AUTHOR(S)**

Brandon Wyatt

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Naval Postgraduate School
Monterey, CA 93943

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

N/A

**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**

Approved for public release. Distribution is unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(maximum 200 words)*

The Wireless Networks for Industrial Automation-Process Automation (WIA-PA) standard is not well known in North America and is a relatively new industrial control system standard when compared to WirelessHart and ISA100.11A. An evaluation of the WIA-PA standard needs to be conducted by Department of Defense and its affiliates to determine whether its operation is on par with WirelessHart and ISA100.11A. The objective of this thesis is to provide a performance analysis of the WIA-PA standard. Utilizing MATLAB, we implemented a custom-built WIA-PA system model and measured the end-to-end delay, and received packet error rate and timeslot utilization. We expect WIA-PA to perform as well as WirelessHart and ISA100.11A in multiple network scenarios. We also found that due to the limitations of MATLAB, further analysis of the standard should be conducted on a network simulator such that network traffic can be properly emulated and the standard's vulnerabilities can be further assessed.

**14. SUBJECT TERMS**

Cyber, ICS, IEEE 802.15.4, Slotted CSMA/CA, WIA-PA, WSN

**15. NUMBER OF PAGES** 109

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UU |

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

**Approved for public release. Distribution is unlimited.**

**PERFORMANCE ANALYSIS OF WIRELESS NETWORKS FOR INDUSTRIAL AUTOMATION-PROCESS AUTOMATION (WIA-PA)**

Brandon Wyatt
Lieutenant, United States Navy
B.S., University of Utah, 2010

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN ELECTRICAL ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL**
**September 2017**

Approved by: Preetha Thulasiraman
Thesis Advisor

John McEachen
Second Reader

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

The Wireless Networks for Industrial Automation-Process Automation (WIA-PA) standard is not well known in North America and is a relatively new industrial control system standard when compared to WirelessHart and ISA100.11A. An evaluation of the WIA-PA standard needs to be conducted by Department of Defense and its affiliates to determine whether its operation is on par with WirelessHart and ISA100.11A. The objective of this thesis is to provide a performance analysis of the WIA-PA standard. Utilizing MATLAB, we implemented a custom-built WIA-PA system model and measured the end-to-end delay, and received packet error rate and timeslot utilization. We expect WIA-PA to perform as well as WirelessHart and ISA100.11A in multiple network scenarios. We also found that due to the limitations of MATLAB, further analysis of the standard should be conducted on a network simulator such that network traffic can be properly emulated and the standard's vulnerabilities can be further assessed.

THIS PAGE INTENTIONALLY LEFT BLANK

# Table of Contents

# List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

# List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

# List of Acronyms and Abbreviations

| | |
|---|---|
| **AL** | application layer |
| **CAP** | contention access period |
| **CCA** | clear channel assessment |
| **CFP** | contention free period |
| **CSMA/CA** | carrier sense multiple access with collision avoidance |
| **CPU** | central processing unit |
| **DCS** | distributed control systems |
| **DLL** | data link layer |
| **DOD** | Department of Defense |
| **FFD** | full function device |
| **GTS** | guaranteed time slot |
| **ICS** | industrial control system |
| **IEC** | International Electrotechnical Commission |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **ISA** | International Society of Automation |
| **ISM** | industrial, scientific, and medical |
| **LR-WPAN** | Low-Rate Wireless Personal Area Networks |
| **MAC** | medium access control |
| **MPDU** | MAC protocol data unit |

| | |
|---|---|
| **NIST** | National Institute of Standards and Technology |
| **NL** | network layer |
| **PAN** | personal area network |
| **PHY** | physical |
| **PPDU** | physical protocol data unit |
| **QoS** | quality of service |
| **RDC** | radio duty cycle |
| **RFD** | reduced function device |
| **SCADA** | supervisory control and data acquisition |
| **SFD** | start-of-frame-delimiter |
| **TDMA** | time division multiple access |
| **USN** | U.S. Navy |
| **WIA-PA** | Wireless Networks for Industrial Automation - Process Automation |
| **WMDSR** | WIA-PA Multi-path Dynamic Source Routing |

# Acknowledgments

I would like to take this opportunity to thank the staff and faculty of NPS for a great educational experience. As Nelson Mandela once said,“Education is the most powerful weapon which you can use to change the world.” Continue the great work as our military leans on you to shape its future leaders.

To Dr. Preetha Thulasiraman, my advisor, thank you for your guidance and mentorship. The project was tough, the road was long, but with your help I made it through. To Dr. John McEachen, my second reader, thank you for your insight and helping make this thesis valuable.

To the staff and faculty within the ECE department, you were able to turn this lowly mathematician into an electrical engineer. I know it was no small feat. Thank you for your dedication to the students and ensuring the quality of education is second to none.

To my children, you inspire me every day to become better. I hope that one day you understand how important you were in helping me attain this achievement.

Finally, to my wife, I could not have done this without you. You have sacrificed so much over the course of my career. I hope that you take pride and ownership of this accomplishment as it could not have been done without you. It is truly a sign of your persistence and dedication as much as it is mine. Thank you!

THIS PAGE INTENTIONALLY LEFT BLANK

# CHAPTER 1:
# Introduction

An industrial control system (ICS) is a broad term that encompasses supervisory control and data acquisition (SCADA), distributed control systems (DCS), and other control systems that work together to meet industrial objectives [1]. According to the National Institute of Standards and Technology (NIST), ICSs "are typically used in electrical, water and wastewater, oil and natural gas, chemical, transportation, pharmaceutical, pulp and paper, food and beverage, and discrete manufacturing (e.g., automotive, aerospace, and durable goods) industries" [1]. These examples are indicative of the many different applications of ICSs in the public and private sectors. As such, ICSs have become an integral part of the U.S. cyber infrastructure.

The Department of Defense (DOD) is one the largest consumers of ICSs in the federal government. The DOD has more than 2.5 million unique ICSs [2]. This collection of specialized systems is pervasive throughout the DOD's infrastructure. The adoption of the ICS has modernized DOD capabilities and helped automate complex processes. The U.S. Navy (USN), both ashore and afloat, utilizes the vast capabilities of the ICS to help "reduce facility maintenance costs, reduce energy consumption, and support mission assurance" [3].

Early control of industrial processes was maintained through either manual control or the utilization of hydraulic systems [4]. As technology has advanced, these manual control processes have been replaced with electronically controlled processes. In traditional ICS networks, sensors are wired, which restricts their use [5]. Over the last several years, control systems have begun to integrate wireless sensor network technology. The use of wireless sensor networks (WSNs) in ICSs provides greater flexibility, scalability, and inherent intelligent processing [6]. They also provide the ability to automate and monitor processes that are either too costly or for which it is too difficult to have wiring installed.

## 1.1 Wireless Sensor Networks

Wireless sensor networks are composed of autonomous sensor networks that are used for acquiring data from the physical environment [6]. The sensor devices that compose these

networks are usually small in size, have limited computing power, use batteries for power, and have an operating range of approximately 10 meters [7]. The main disadvantage of a sensor device is the power constraint. The limited power range of a sensor makes the WSN incompatible with standard Wi-Fi protocols. This constraint has driven the development of new wireless standards for Low-Rate Wireless Personal Area Networks (LR-WPAN).

The Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 standard for LR-WPAN was released in 2003. IEEE 802.15.4 defines the specifications for the physical (PHY) and medium access control (MAC) layers of devices that require low battery consumption and have low data rates [7]. There are several different standards that aim to meet the requirements of WSNs within an ICS, including WirelessHart, International Society of Automation (ISA)100.11A, and Wireless Networks for Industrial Automation - Process Automation (WIA-PA) [8]. These standards use the PHY/MAC layers specified in IEEE 802.15.4 and then build proprietary data link layer (DLL), network layer (NL), and application layer (AL) protocols on top.

## 1.2 Research Motivations and Objectives

Two of the most prevalent standards in WSNs for an ICS are WirelessHart and ISA100.11A [9]. Both standards are internationally accepted and have been thoroughly researched, as is documented in the literature. Major international companies such as Honeywell, Siemens, and Emerson have developed wireless instrumentation technology based on WirelessHart and ISA100.11A.

The WIA-PA standard was first proposed by the Chinese Industrial Wireless Alliance in 2007. It was approved by the International Electrotechnical Commission (IEC) as an international standard (IEC62601Ed.1.0.) in 2011 and is currently in its second revision (IEC62601:2016). WIA-PA offers various advantages over WirelessHart and ISA100.11A. WIA-PA adopts adaptive frequency hopping, two-level aggregation, and utilization of a mesh-star network topology [10]. WIA-PA utilizes beacons and is able to coexist with other beacon-enabled networks, while WirelessHart and ISA100.11A are unable to send or understand beacon messages and therefore cannot coexist with beacon-enabled networks [9]. WIA-PA's mesh and star topology allows distributed communication allocation [9]. The distributed allocation is completed by the network manager. The network manager allocates

resources to the cluster heads, and then the cluster heads allocate resources to the devices within their clusters. This is in contrast to WirelessHart where the network manager allocates resources to all devices. ISA100.11A utilizes a backbone of routers for centralized management [9]. WIA-PA also implements static routing with redundant routing devices to meet the real-time requirements of an ICS, as compared to source and graph routing in WirelessHart and ISA100.11A. Graph routing increases the latency of the network; therefore, the routing algorithms used in WirelessHart and ISA100.11A must be optimized specifically for use with these standards [9].

Despite these advantages, the WIA-PA standard is not well known in North America and is a relatively new ICS standard when compared to WirelessHart and ISA100.11A. Although it is part of the IEC, there is not an extensive amount of research on WIA-PA documented in the literature. The limited research that is available has been conducted by Chinese research institutions. Given that the use cases for WIA-PA are extensive [11], an evaluation of the WIA-PA standard needs to be conducted by DOD and its affiliates to determine whether its operation is on par with WirelessHart and ISA100.11A. In addition, an analysis of WIA-PA would provide further insight into possible future exploitation and vulnerability assessment of the standard.

## 1.3 Thesis Contributions

The objective of this thesis is to provide a performance analysis of the WIA-PA standard. The contributions of this thesis are as follows:

- Performance evaluation of WIA-PA on a custom-built system model implemented in MATLAB
- Analysis of results based on the following WIA-PA performance metrics: end-to-end delay, received packet error rate, and timeslot utilization

## 1.4 Thesis Organization

The remainder of this thesis is organized as follows: In Chapter 2 we present a review of related research involving WIA-PA, WirelessHart, and ISA100.11A. The IEEE 802.15.4 protocol and the WIA-PA standard are discussed in Chapter 3. The experimental model and analysis of WIA-PA network performance is provided in Chapter 4. In Chapter 5 we

present our conclusions and recommendations for future work. Appendix B contains all the MATLAB code used for the simulations.

# CHAPTER 2:
# Research Review

As technology advances and WSNs begin to be implemented in ICSs, research has been executed to develop and test methods to meet the rigorous requirements of real-time industrial processes. In this chapter we review some of the research that has been conducted for each of the three international standards for ICS: WirelessHart, ISA100.11A, and WIA-PA.

## 2.1 WirelessHart

In 2008, the IEC approved WirelessHart as an international standard. It is an extension of the Highway Addressable Remote Transducer (HART) protocol, which is a wired ICS protocol. WirelessHart utilizes graph routing, but leaves the scheduling to be implemented by the user. As discussed in Section 1.2, graph routing increases the latency of the network. To minimize this latency, the authors of [8], [12], and [13] proposed multiple scheduling algorithms to find an optimization that is feasible with the energy constraints of WSNs. In [14], the authors proposed the implementation of a forward error correction (FEC) scheme, the employment of channel polarization diversity, and a reduction of the WirelessHart packet header size. The authors demonstrated that these changes reduced the power consumption of the protocol and increased energy efficiency.

Since WirelessHart utilizes the non-beacon-enabled mode of IEEE 802.15.4 and time division multiple access (TDMA), synchronization across the network is of utmost importance. Even a slight disparity in timing can have grave consequences in the industrial environment. The authors of [15] introduced mathematical models to enhance the study of time synchronization for WirelessHart, while in [16] and [17], the authors proposed methods to improve the synchronization precision of the network. To help with testing and studying WirelessHart, several researchers have offered different simulation methods [18], [19]. It is important to have multiple ways to evaluate the different standards to ensure their reliability.

## 2.2 ISA100.11A

ISA100.11A is a standard that was developed in 2009 and gained IEC approval in 2014. As with the other WSN standards, ISA100.11A utilizes the 2.4 GHz band of the industrial, scientific, and medical (ISM) unlicensed band. IEEE 802.11 uses the same band. The authors of [20] showed that an ISA100.11A network could operate without degradation within radio range of an IEEE 802.11 network. There are many protocols that operate within the 2.4 GHz band, which leads to congestion and interference when multiple protocols are operating within radio range. To assist with operating in a congested frequency band, [21] and [22] studied adaptive channel diversity for ISA100.11A and showed that this method helps to avoid interference from congestion.

ISA100.11A, as in WirelessHart, utilizes the non-beacon-mode of IEEE 802.15.4 and a timeslot scheme for communication. With this comes the need for synchronization and a scheduling algorithm to ensure real-time processing. The authors of [23] offered a Traffic-Aware Message Scheduling method that improved throughput and end-to-end delay for ISA100.11A. As for synchronization, [24] proposed a novel self-organized synchronization method based on the synchronization patterns of male fireflies found in Southeast Asia.

## 2.3 WIA-PA

The Chinese-developed WSN standard for ICS, WIA-PA, was accepted by the IEC in 2011 [10]. This standard is the only one of the three that implements the beacon-enabled mode of the IEEE 802.15.4 standard. The beacons are used for initial synchronization of joining nodes, but the standard also utilizes a time synchronization packet that was proposed in [25]. By using beacons, WIA-PA is compatible with other beacon networks, as discussed in [9]. This allows WIA-PA to operate in areas without causing or experiencing interference. In addition, WIA-PA is the only standard of the three that fully implements IEEE 802.15.4 [9]. This allows WIA-PA interoperability with other WSNs. The authors in [26] noted that WIA-PA's use in ICSs is limited due to real-time constraints. They proposed a scheduling algorithm that utilized the centralized and decentralized communication allocation properties of the standard to achieve real-time requirements, thereby increasing the use cases of the standard.

WIA-PA utilizes static routing by default. In [27], a dynamic source routing protocol called

WIA-PA Multi-path Dynamic Source Routing (WMDSR) was proposed. As noted in [9], source routing induces a higher latency as compared to static routes. Therefore, source routing was not adopted by the standard.

## 2.4 Chapter Summary

In this chapter, we discussed the existing research pertaining to WirelessHart, ISA100.11A, and WIA-PA. We presented the limitations and challenges of each standard. We also discussed the advantages of WIA-PA.

THIS PAGE INTENTIONALLY LEFT BLANK

# CHAPTER 3:
# WIA-PA System Model Implementation

In the following sections of this chapter we discuss the IEEE 802.15.4 standard on which WIA-PA is based. We also discuss the ways in which WIA-PA uses and extends IEEE 802.15.4.

## 3.1 IEEE 802.15.4

As was stated in Chapter 1, IEEE 802.15.4 defines the PHY and MAC layers of lower power devices. In the sections that follow, we describe the components of IEEE 802.15.4 that are implemented for our WIA-PA model.

### 3.1.1 Components

The IEEE 802.15.4 protocol defines two types of devices, a full function device (FFD) and a reduced function device (RFD) [7]. The FFD refers to a device that has routing capability. An FFD can act as a personal area network (PAN) coordinator, a coordinator, or a device within the network. An FFD acting as a PAN coordinator has ownership of the network and allows other devices to join. If the FFD is acting as a coordinator, it is routing traffic to and from devices and the PAN coordinator, but does not allow for devices to join the network. If the FFD is set to operate in the device mode, it will only transmit data and does not have any routing functionality or joining capability. An RFD is generally a sensor node that is providing data to the network. It does not have routing capability or ownership of the network.

### 3.1.2 Topology

Within an IEEE 802.15.4-enabled network, these FFDs and RFDs can be combined to form either a star network topology or a peer-to-peer network topology [7]. The star topology is a one-hop network construct in which there is a single PAN coordinator and all other devices of the network can only communicate with the PAN coordinator. The next topology is peer-to-peer or mesh. In this network construct there is only one PAN coordinator but

multiple coordinators and each device in the network can communicate with any other device. This type of network is considered multi-hop, as the data can traverse multiple devices from source to destination. Examples of the star and peer-to-peer topologies are shown in Figure 3.1.

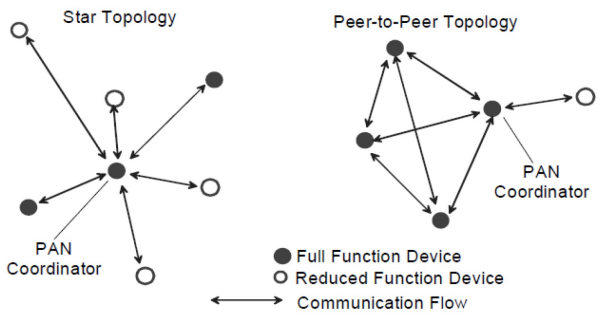


Figure 3.1. Examples of Star and Peer-to-Peer Network Topologies. Source: [7].

A cluster-tree network topology is a combination of the star and peer-to-peer topologies and is the topology used in this thesis. In this construct there is one PAN coordinator and multiple coordinators. The coordinators form a peer-to-peer network. The devices connect to the coordinators to form the star topologies. Note that the coordinators can communicate with other coordinators in range. However, the devices can only communicate with their corresponding coordinator. An example of a cluster-tree network topology is shown in Figure 3.2.

### 3.1.3 Physical Layer

The physical layer of the IEEE 802.15.4 utilizes several operational frequencies bands with differing data rates [7]. Listed in Table 3.1 are all available bands for IEEE 802.15.4 as well as the corresponding data rates and modulation schemes. The focus of this thesis will be on the 2.4 GHz frequency band. In this frequency band, 16 channels are available between 2.4 and 2.4835 GHz with 5 MHz spacing. The physical layer is also responsible for the radio duty cycle (RDC) and clear channel assessment (CCA). This RDC turns the radio on and off and is important to meet the low power constraint of the sensor devices. The CCA is a process by which the physical layer determines if the channel is in use.

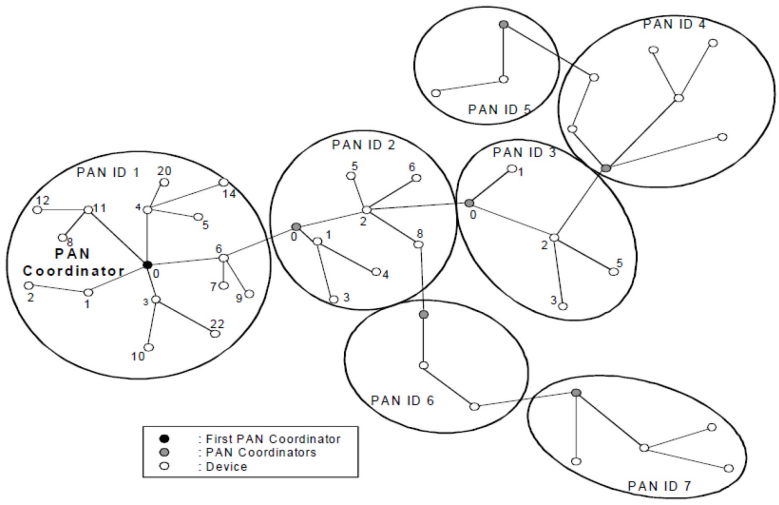


Figure 3.2. Example of a Cluster-Tree Network Topology. Source: [7].

**PHY Protocol Data Unit**

The physical protocol data unit (PPDU) has a maximum size of 133 bytes. It includes a preamble, composed of 32 zeros, and an 8-bit start-of-frame-delimiter (SFD), which is defined as 11100101 [7]. The PPDU also contains the payload and a seven-bit field representing the length of the payload in bytes. The length field limits the PPDU a maximum payload of 127 bytes. The PPDU for IEEE 802.15.4 is presented in Figure 3.3.

| | | Octets | | |
|---|---|---|---|---|
| | | 1 | | variable |
| Preamble | SFD | Frame length (7 bits) | Reserved (1 bit) | PSDU |
| SHR | | PHR | | PHY payload |

Figure 3.3. PPDU for IEEE 802.15.4. Source: [7].

### 3.1.4 Medium Access Control Layer

The MAC layer is the interface between the physical layer of IEEE 802.15.4 and the higher layer protocols. It is responsible for managing beacons, accessing the channel, managing time slot allocation, as well as device association and disassociation [7]. The IEEE 802.15.4 MAC layer supports two modes: beacon-enabled and non-beacon-enabled. In the beacon-enabled mode, the PAN transmits beacons to synchronize all attached devices and provide

Table 3.1. Frequency Bands and Data Rates for IEEE 802.15.4. Source: [7].

| PHY (MHz) | Frequency band (MHz) | Spreading parameters | | Data parameters | | |
|---|---|---|---|---|---|---|
| | | Chip rate (kchip/s) | Modulation | Bit rate (kb/s) | Symbol rate (ksymbol/s) | Symbols |
| 780 | 779–787 | 1000 | O-QPSK | 250 | 62.5 | 16-ary orthogonal |
| 780 | 779–787 | 1000 | MPSK | 250 | 62.5 | 16-ary orthogonal |
| 868/915 | 868–868.6 | 300 | BPSK | 20 | 20 | Binary |
| | 902–928 | 600 | BPSK | 40 | 40 | Binary |
| 868/915 (optional) | 868–868.6 | 400 | ASK | 250 | 12.5 | 20-bit PSSS |
| | 902–928 | 1600 | ASK | 250 | 50 | 5-bit PSSS |
| 868/915 (optional) | 868–868.6 | 400 | O-QPSK | 100 | 25 | 16-ary orthogonal |
| | 902–928 | 1000 | O-QPSK | 250 | 62.5 | 16-ary orthogonal |
| 950 | 950–956 | — | GFSK | 100 | 100 | Binary |
| 950 | 950–956 | 300 | BPSK | 20 | 20 | Binary |
| 2450 DSSS | 2400–2483.5 | 2000 | O-QPSK | 250 | 62.5 | 16-ary orthogonal |
| UWB sub-gigahertz (optional) | 250–750 | As defined in 14.4.1 | | | | |
| 2450 CSS (optional) | 2400–2483.5 | As defined in 13.2 | | 250 | 167 (as defined in 13.4.2) | |
| | | As defined in 13.2 | | 1000 | 167 (as defined in 13.4.2) | |
| UWB low band (optional) | 3244–4742 | As defined in 14.4.1 | | | | |
| UWB high band (optional) | 5944–10 234 | As defined in 14.4.1 | | | | |

the structure of the superframe. The superframe structure is shown in Figure 3.4. The superframe provides the basis for communication within the PAN.

The rate at which beacons are sent is the reciprocal of the beacon interval $BI$, is determined by the variable beacon order $BO$ and can be changed to meet the demands of the PAN. Each coordinator, or FFD, within the PAN will use the same $BI$. The $BO$ is a value within the range $0 \leq BO \leq 15$. If the $BO = 15$, the PAN is operating in the non-beacon-enabled

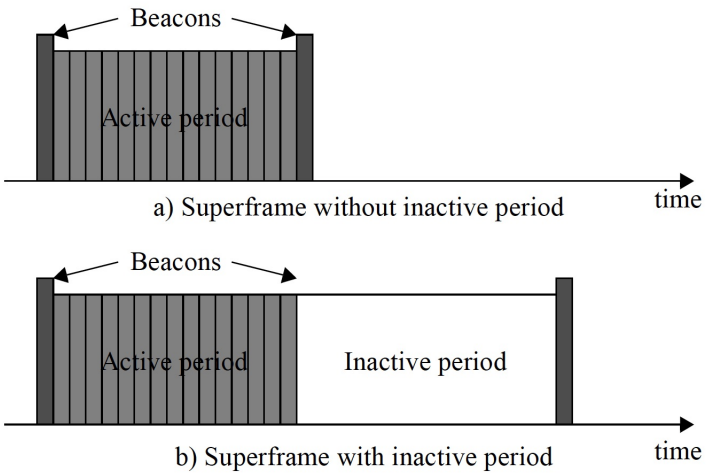


Figure 3.4. Superframe Structure for IEEE 802.15.4. Source: [7].

mode and will not transmit beacons regularly. Otherwise,

$$BI = aBaseSuperframeDuration \cdot 2^{BO}, \tag{3.1}$$

where $aBaseSuperframeDuration$ is defined as

$$aBaseSuperframeDuration = aNumSuperframeSlots \cdot aBaseSlotDuration \tag{3.2}$$

with $aBaseSlotDuration = 60$ symbols/slot and $aNumSuperframeSlots = 16$ slots.

The $BO$ establishes the frequency of the beacon interval, but the duration of the active period is governed by the superframe order $SO$. This value is within the range $0 \leq SO \leq BO \leq 15$. If the $SO = 15$, then $BO = 15$ and the PAN is operating in the non-beacon-enabled mode. For the superframe to contain an inactive period, $SO < BO$. An example of the superframe inactive period is shown in Figure 3.4b. The superframe duration, $SD$, is calculated as

$$SD = aBaseSuperframeDuration \cdot 2^{SO}. \tag{3.3}$$

The active period of the superframe is divided into 16 time slots of equal duration. These time slots are grouped into a contention access period (CAP) and a contention free period (CFP). During the CAP, if a device needs to transmit it must use the slotted variant of

carrier sense multiple access with collision avoidance (CSMA/CA) prior to transmission. CSMA/CA is described in further detail in Section 3.1.5.

Devices requiring a guaranteed quality of service (QoS) are assigned a guaranteed time slot (GTS) during the CFP. This means that a specific timeslot has been allocated to a specific device similar to a TDMA construct. The CAP is shortened by the amount of the CFP so that both periods are contained within the active portion of the superframe. The inactive period of the superframe allows devices to sleep, meaning that a device has the ability to turn off its radio with the assumption that no communications will occur during this period. As mentioned previously, the IEEE 802.15.4 non-beacon-mode does not send beacons regularly and all devices desiring to transmit must contend for the channel using the unslotted CSMA/CA without the option of a GTS. The slotted variant of CSMA/CA will be the focus of this thesis.

**MAC Protocol Data Unit**

The MAC protocol data unit (MPDU) sets the structure for the MAC frame. The general MAC frame construct is shown in Figure 3.5.

<table>
<tr><td>Octets: 2</td><td>1</td><td>0/2</td><td>0/2/8</td><td>0/2</td><td>0/2/8</td><td>0/5/6/10/14</td><td>variable</td><td>2</td></tr>
<tr><td rowspan="2">Frame Control</td><td rowspan="2">Sequence Number</td><td>Destination PAN Identifier</td><td>Destination Address</td><td>Source PAN Identifier</td><td>Source Address</td><td rowspan="2">Auxiliary Security Header</td><td rowspan="2">Frame Payload</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="5">Addressing fields</td></tr>
<tr><td colspan="7">MHR</td><td>MAC Payload</td><td>MFR</td></tr>
</table>

Figure 3.5. General MAC Frame Structure. Source: [7].

Each frame uses the frame control field to identify how the frame is constructed [7]. The details of the frame control field can be seen in Figure 3.6. The frame type field identifies which frame is being transmitted: beacon, data, acknowledgement, or command. The security enabled field indicates whether or not the PAN is operating with encryption. The frame pending field is used by a coordinator to signal that data is pending for a node within its cluster. The PAN ID compression field is a one-bit field that allows for the omission of one of the two PAN identifier fields in the MAC frame, if the two fields are the same. The

addressing modes are two-bit fields that identify whether the corresponding address field is present, and if it contains the 16-bit or 64-bit address.

| **Bits: 0–2** | **3** | **4** | **5** | **6** | **7–9** | **10–11** | **12–13** | **14–15** |
|---|---|---|---|---|---|---|---|---|
| Frame Type | Security Enabled | Frame Pending | AR | PAN ID Compression | Reserved | Dest. Addressing Mode | Frame Version | Source Addressing Mode |

Figure 3.6. Frame Control Field Structure. Source: [7].

IEEE 802.15.4 defines four MAC frame types: beacon, acknowledgement, data, and command. The beacon frame shown in Figure 3.7 is used to synchronize the devices in the PAN and to provide the structure of the superframe. The MAC payload of the beacon frame contains the superframe specifications, shown in Figure 3.8. The devices within the PAN extract the superframe specifications to determine the active and inactive portions, as discussed in Section 3.1.4.

| **Octets: 2** | **1** | **4/10** | **0/5/6/10/14** | **2** | **variable** | **variable** | **variable** | **2** |
|---|---|---|---|---|---|---|---|---|
| Frame Control | Sequence Number | Addressing fields | Auxiliary Security Header | Superframe Specification | GTS fields | Pending address fields | Beacon Payload | FCS |
| MHR | | | | MAC Payload | | | | MFR |

Figure 3.7. Beacon Frame. Source: [7].

Embedded within the superframe specification is the *BO*, *SO*, the timeslot in which the CAP ends and whether or not the beacon received is from the PAN coordinator.

| **Bits: 0–3** | **4–7** | **8–11** | **12** | **13** | **14** | **15** |
|---|---|---|---|---|---|---|
| Beacon Order | Superframe Order | Final CAP Slot | Battery Life Extension (BLE) | Reserved | PAN Coordinator | Association Permit |

Figure 3.8. Superframe Specification. Source: [7].

Acknowledgement frames within IEEE 802.15.4 are only transmitted after receipt of data or command frames and if the acknowledgement request field is set in the frame control field.

| Octets: 2 | 1 | 2 |
|---|---|---|
| Frame Control | Sequence Number | FCS |
| MHR | | MFR |

Figure 3.9. Acknowledgement Frame. Source: [7].

The acknowledgement frame shown in Figure 3.9, contains no addressing information and only identifies the frame sequence number to which it is in response.

The command frame structure can be seen in Figure 3.10. These frames are used for control of the PAN, to include device joining, association, disassociation, and for devices to request pending data from the coordinator.

| Octets: 2 | 1 | variable | 0/5/6/10/14 | 1 | variable | 2 |
|---|---|---|---|---|---|---|
| Frame Control | Sequence Number | Addressing fields | Auxiliary Security Header | Command Frame Identifier | Command Payload | FCS |
| MHR | | | | MAC Payload | | MFR |

Figure 3.10. Command Frame. Source: [7].

The data frames are used to communicate all other data not related to coordination and control of the PAN. The data frame structure is shown in Figure 3.11.

| Octets: 2 | 1 | variable | 0/5/6/10/14 | variable | 2 |
|---|---|---|---|---|---|
| Frame Control | Sequence Number | Addressing fields | Auxiliary Security Header | Data Payload | FCS |
| MHR | | | | MAC Payload | MFR |

Figure 3.11. Data Frame. Source: [7].

### 3.1.5 Functional Overview

#### Data Transfer

There are three data transfer concepts in IEEE 802.15.4 [7]. Data can be transmitted from a coordinator, to a coordinator, and peer-to-peer. When a coordinator needs to transmit data to

a connected device and is operating within a beacon-enabled PAN, it will utilize the beacon to signal that it has data and indicate the device for which the data is intended. If the PAN is non-beacon-enabled, the coordinator will hold the data until the intended device submits a request for the data. When data is transferred to a coordinator in the beacon-enabled mode, the device must synchronize to the beacon frame and transmit when appropriate. In a non-beacon-enabled PAN, the device utilizes unslotted CSMA/CA to transmit the data. For data to be transferred among peers, the devices transmit directly to all other devices within its radio transmit range. This requires each device to either listen continuously or synchronize with each peer to ensure receipt of transmission.

**CSMA/CA**

CSMA/CA is a carrier sensing protocol, by which a device wanting to transmit waits a random amount of time (called a backoff period), senses the channel to determine if it is available, and then transmits when able [7]. The algorithm implemented for CSMA/CA is represented in Appendix A. CSMA/CA can be utilized in one of either two modes, slotted or unslotted. When the unslotted mode of CSMA/CA is used, the backoff period of each device is unrelated to any other device. In the slotted mode, the backoff period boundaries are aligned to the timeslots of the superframe for each device.

## 3.2 WIA-PA

As previously discussed, WIA-PA is based on and extends the IEEE 802.15.4 standard. The following sections describe how WIA-PA does this.

### 3.2.1 Components

WIA-PA maintains the IEEE 802.15.4 definitions of FFD and RFD, but renames the devices based on roles [28]. The standard identifies five devices: host configuration computer, gateway, routing device, field device, and handheld device. The purpose of the host configuration computer is to set the parameters for all of the superframes within the network, provide resources when devices initiate a join request, assign communication resources to the routing devices, and create and disseminate all routing tables within the network. The gateway and routing devices are the same as the FFD in IEEE 802.15.4 and provide the

network connectivity and routing. The field devices are sensors and the handheld devices are remote network management or user access points.

### 3.2.2 Topology

The topology that WIA-PA utilizes is a combination peer-to-peer and star [28]. The routing devices communicate with each other in a peer-to-peer topology, with each routing device able to head its own star. An example of this network topology is shown in Figure 3.12.

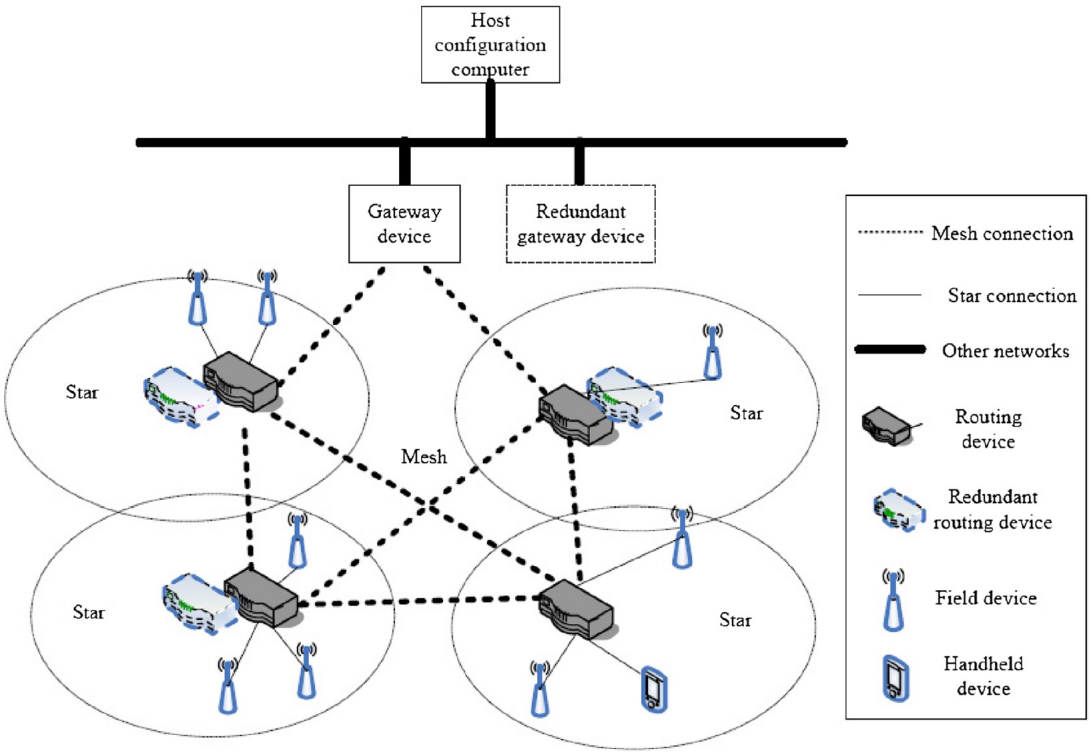


Figure 3.12. Example of a WIA-PA Network Topology. Source: [28].

### 3.2.3 Physical Layer

WIA-PA only allows the 2.4 GHz frequency band and a data rate of 250 kbps [28]. IEEE 802.15.4 defines channels 11 - 26 to frequencies 2,405 MHz to 2,480 MHz, with 5 MHz spacing [7]. WIA-PA does not support channel 26, as defined in IEEE 802.15.4, as it is not universally compliant [28].

### 3.2.4 Medium Access Control Layer

The MAC layer specification in WIA-PA utilizes the beacon-enabled mode of IEEE 802.15.4 and the $BI$ is defined as in (3.1) [28]. The active portion of the WIA-PA superframe is divided into the CAP and the CFP. The CAP is only utilized for device joining and association and management of the PAN. CSMA/CA is utilized during this period for communication. Upon completion of the CAP, all other communications occur during assigned timeslots. The CFP is designated for communication between the cluster head and handheld devices. WIA-PA extends the IEEE 802.15.4 superframe and utilizes the inactive portion for intra- and inter-cluster communication as well as device sleeping. As the WIA-PA superframe includes the inactive portion, the base superframe duration is calculated by setting the $aNumSuperframeSlots$ to 32 in (3.2).

WIA-PA allows for multiple superframes across the PAN, however each routing device and RFD is only associated with one superframe at a time [28]. Each superframe can be made up of a different number of timeslots, but the timeslot duration will be constant across the PAN. This is configured by the host configuration computer based on the requirements of the devices within a given cluster. The short address of devices joining the network are also assigned by the host configuration computer. The short address is 16 bits long, where the most significant eight bits designate the cluster and the least significant eight bits identify the device within the cluster.

### 3.2.5 MAC Protocol Data Unit

The MPDU has some differences from that of IEEE 802.15.4. Within the frame control field, the security field is always zero and the PAN ID compression flag is always set to one [28]. This means that communication is not allowed between PANs. After a device joins, it must utilize its short address to communicate. WIA-PA does not utilize the GTS of IEEE 802.15.4. Instead, the network layer assigns links and communication resources to devices based on the update rate of the device. The link information includes a timeslot, a channel, and type of communication, which is either transmit, receive, or share.

The MAC layer is also responsible for synchronizing devices upon joining the network. This occurs via the beacon frame. WIA-PA adds a payload to the beacon that includes the cluster ID with which the beacon is associated, the absolute slot number, a timer value, and

the channel on which the next beacon will be transmitted [28]. The absolute slot number is a timeslot counter that is incremented by one for each timeslot. The timer value is the elapsed time from the transmission of the beacon to the beginning of the timeslot. Before a device joins the network, it passively scans for beacons and synchronizes with the information in the beacon payload. This synchronization does not have enough fidelity to meet real-time requirements so the routing devices will also transmit synchronization command frames.

## 3.3 Chapter Summary

In this chapter, we discussed the IEEE 802.15.4 standard and the ways in which WIA-PA uses and extends the standard. The discussion of this chapter provides the foundation for our MATLAB implementation of WIA-PA.

# CHAPTER 4:
# Results and Analysis

In this chapter we present our experimental design and system model used in the simulations. We also discuss the various components of the WIA-PA standard that are implemented for this thesis. The performance metrics that we evaluated on the WIA-PA model are as follows:

- end-to-end delay: the delay incurred from the time a packet is transmitted by a node to when it is reached by the gateway
- received packet error rate: the number of packets received in error by the gateway calculated as a percentage of total packets transmitted
- timeslot utilization: a measure of how well WIA-PA nodes maximize timeslot use

## 4.1 Experimental Design

**Initialization**

The system model includes two MATLAB class definitions: wia_ffd.m (Appendix B.1.1) and wia_rfd.m (Appendix B.1.2). These classes form the template for the devices and allocate properties necessary for the devices to operate in the network. For example, the classes create buffers $rx$ and $tx$, which are buffers to hold data that either a device received or will transmit. Other properties that are created, such as $cluster\_id$, $my\_short\_address$, and $bcn\_payload$, provide allocated resources for the device to store data from communications it receives from the network. The $cluster\_id$ identifies the cluster to which a device is connected and also serves as the first two octets of the device's short address once associated. The short address of a device is stored in the $my\_short\_address$ property.

The beacons that are transmitted by FFDs contain payload information that is important for network synchronization [28]. This information is stored in the $bcn\_payload$ property. When a device is initialized, certain properties are set depending on the type of device desired. For example, when an FFD is initialized as a PAN coordinator, the $associated$ parameter, the $my\_short\_address$ property, and information required for beacon structure are set. The other FFDs and RFDs are initialized without this information and will obtain

the appropriate property values during the association process. Along with allocating these properties, these classes build timers to allow for the device functions. The MATLAB timer class was utilized to allow for pseudo multi-threading capability. The timers themselves will run as an individual process, but the functions that the timers execute are scheduled to the processor serially. These timers allow the device to send data or check the status of properties on a given interval. The main difference between the two class definitions is that the wia_ffd class uses a beacon timer, while the wia_rfd class uses a data timer. In our model, we are not considering the FFD as sending its own data, even though it is able to. The RFDs, however, do not transmit beacons and, therefore, do not have a beacon timer [28]. The flow chart describing device initialization is shown in Figure 4.1.

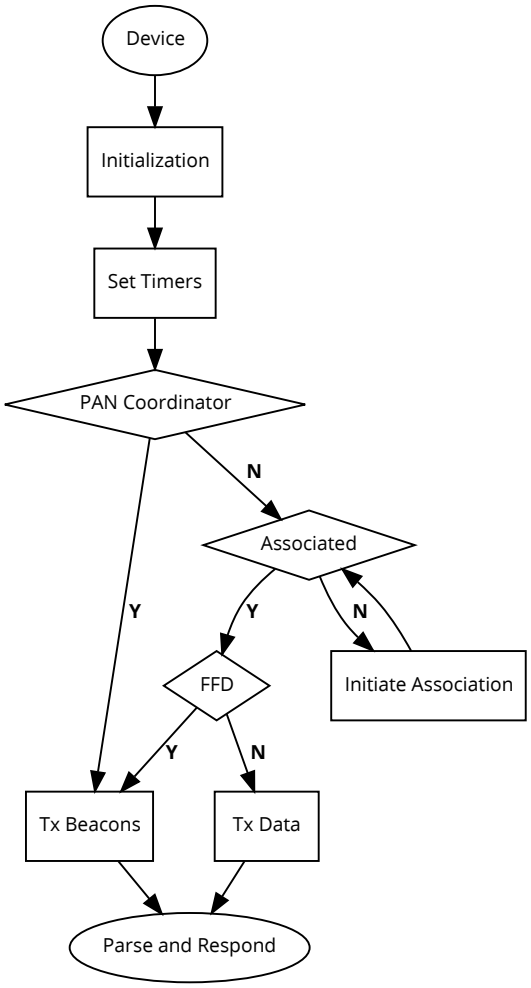


Figure 4.1. Device Initialization Flow Graph

**Association**

When a device (that is not the PAN coordinator) is initialized, it is not associated to the PAN. The device begins listening for beacons. Once a beacon is received, the device will transmit an association request command frame within the CAP. Once the PAN coordinator receives the association request, from one or multiple devices, it will set the $framepending$ flag in the beacon and append the intended addresses in the beacon payload. After receiving the next beacon, each device listed in the payload will send a data request to poll for the pending data. The PAN coordinator will then provide the device with a short address for the network [28].

If the associating device is a routing device, its address will be $XX00$ where $XX$ is a hexadecimal representation of the cluster ID [28]. If the device is an RFD, its address will be $XXYY$, where $XX$ is the cluster ID and $YY$ is the host identification within the cluster. Once a routing device is associated, it will begin transmitting its own beacons to devices within its transmission range. The PAN coordinator has to provide the association response. For a node two or more hops away, the parent must forward the association request to the PAN coordinator and forward the response back to the node. No device is able to send data on the network until it is associated. Once associated, the node sends data to its parent, which forwards data to the gateway. An example of the association flow can be seen in Figure 4.2 [28].

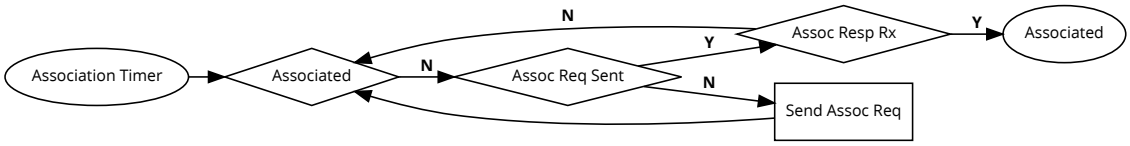


Figure 4.2. Association Flow Graph

**Data Transfer**

To allow for the simulation of data transfer, we build neighbor tables to simulate the radio transmission range of the devices. These neighbor tables are created after all the devices are initialized, but before the timers are started. When a device wishes to communicate, it calls the transmit function (Appendix B.2.28). This is the function that fully creates the PPDU that will be transmitted. Within the function is a call to have the MPDU created (Appendix B.2.12). When the MPDU is returned to the transmit function, the rest of the PPDU header

is added and a call to the channel (Appendix B.2.8) function is made. The transmit function then adds the packet to a send buffer. Each device has a send timer that polls this buffer and determines when a packet is available for transmission. If the packet is a data frame, the packet will be scheduled to be added to the channel during the appropriate timeslot, while a command frame will need to be transmitted during a CAP [28]. The channel function copies the data to be transmitted into the channel buffer of each device within its neighbor table. To simulate data propagation within the channel, we delay copying the data to the channel buffer by the amount of time calculated as

$$Delay = \frac{length(PPDU)}{R_b}, \tag{4.1}$$

where $R_b$ is the bit rate of 250 kbps [28]. A depiction of how the transmit function operates can be seen in Figure 4.3.

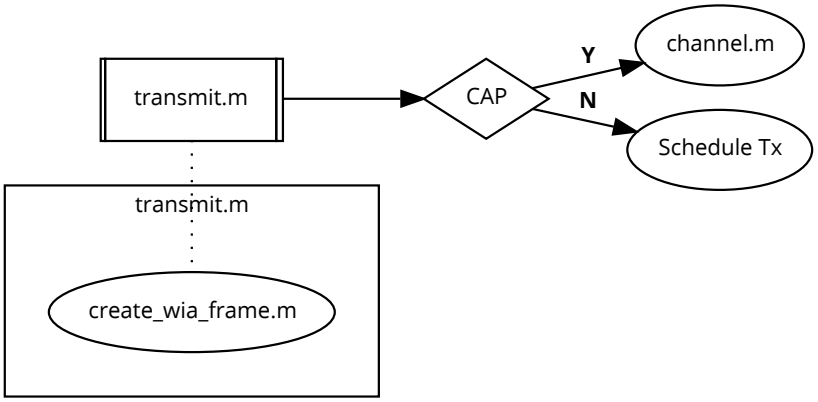


Figure 4.3. Transmit Flow Graph

Each device also polls its receive buffer, utilizing the parse timer to see if data is available. When data is found, the devices immediately parse the packet to build the required information for the superframe. The parse timer calls the parse function (Appendix B.2.19), which simply removes the first item from a received frame buffer, calls the $parse\_wia$ function, and fills the $rx$ buffer as the data is parsed. Once the received packet has been parsed, the $parsed$ function is called. The $parsed$ function determines how the device responds depending on how its parameters are currently set. For example, after a device receives a beacon, it is the $parsed$ function that checks to see if the device is associated or whether it

is an FFD. It then initiates the appropriate response. The parse timer continuously checks the state of the receive buffer. A simple flow chart depicting the use of this timer is shown in Figure 4.4.

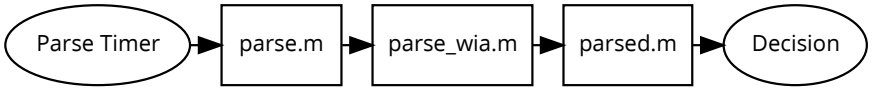


Figure 4.4. Parse Timer Flow Graph

## 4.2 Simulation Model

WIA-PA allows for the number of timeslots within a superframe to be set by the user [28]. For simplification, we chose to use a superframe consisting of 32, 10-ms timeslots. We divided the 32 timeslots equally among the four periods of the superframe: CAP, CFP, intra-cluster period, and inter-cluster period. We ignored a dedicated sleep period across the network. Within the inter-cluster period, we required the cluster heads to share a single channel. WIA-PA allows for multiple channel use, but utilizing multiple channels reduces the problem to a single cluster scenario [28]. We chose to utilize only a single channel within the inter-cluster period to expand our testbed. By using the single channel during the inter-cluster period, we are limited to eight timeslots for each period of the superframe, which restricts our network size to a total of eight nodes. The number of nodes is determined by the number of timeslots available in the inter-cluster period. Since there are only eight timeslots, the cluster heads are only able to forward data to the gateway for eight nodes.

Next, we assumed that each device has successfully joined the network, in accordance with [28]. Our model does not simulate the joining process because we were not able to use multiple processing threads in our MATLAB implementation. The utilization of multiple process threads allows for processes to be scheduled in parallel. Without using multiple processing threads, each process is scheduled serially on the central processing unit (CPU) instead of in parallel.

During the joining process, a node is allocated a short address and a timeslot for data communication [28]. The timeslot allocated to each node, within the intra-cluster period, corresponds to the node's short address. For example, the node with short address 0101 is the first node of cluster 01. It will transmit at the start of the first timeslot of the intra-cluster

period, or timeslot 16 within the superframe [28]. The cluster heads transmit in order of cluster number and use the number of slots equal to the number of child nodes. Cluster head 0100 with three nodes would then transmit for the first three timeslots of the inter-cluster period, or timeslots 24 to 26. We implemented this scheduling method, since a scheduling method is not defined in [28].

Another assumption that we made is that data is transmitted only from the node to the gateway. During the delay and received packet error rate simulations, we did not include acknowledgement frames, since acknowledgement frames are not required [28]. Acknowledgement frames were, however, considered during the timeslot utilization calculations.

As discussed previously, our implementation does not utilize multiple process threads. MATLAB is only able to schedule processes at intervals of 1 ms, but implementing this becomes too computationally intensive and other process do not get scheduled. Due to this, we expanded our simulation time to be in terms of seconds, and scaled data rates and processing rates to match. For simulation purposes, our timeslot within the superframe is one second and the data rate is 2.5 kbps, as compared to the 10 ms and 250 kbps intended [28].

## 4.3 End-to-End Delay

In this section we measure the time required for a gateway to receive data from a node within the network. We look at three situations: a single cluster containing a single node, a single cluster containing eight nodes, and two clusters containing four nodes per cluster.

### 4.3.1 Single Cluster Containing Single Node

In the case of a single cluster containing a single node, the theoretical delay $t_{delay}$ is defined as

$$t_{delay} = t_{trans} + t_{prop} + 8t_{slot}, \tag{4.2}$$

where $t_{trans}$ is how long it takes for the device to physically send the data, $t_{prop}$ is how long the data takes to propagate through the channel (based on a distance of 10 m [7]), and $8t_{slot}$ is the duration between the node's allocated timeslot and that of the cluster head [28]. Based on the parameters of our model and with a frame size of 176 bits, $t_{delay}$ equates to 8.07 ms.

We observed an average end-to-end delay of approximately 8.083 ms from 1000 transmitted data frames. The measured delay shows a .16% increase over the expected delay, which is believed to be caused by variances in the MATLAB implementation.

### 4.3.2 Single Cluster Containing Eight Nodes

Our next simulation increased the cluster size. The delay equation (4.2), applies to all nodes within the cluster. With this, we expected to see a delay of 8.07 ms, as in the single node case. The average delay that we observed in this case was 8.125 ms. This delay is a .52% increase over the expected value from the single node case. We believe that the increased delay is due to computational resources in MATLAB. The delay grows incrementally as the number of nodes grows. However, in this scenario, the number of nodes should not affect the delay of any other node.

### 4.3.3 Two Clusters Containing Eight Nodes Total

For our third simulation, we increased the number of clusters, and included four nodes in each cluster. In this case, the expected delay is the 8.07 ms for the nodes in the first cluster, but the second cluster has an additional delay of $4t_{slot}$, as seen in

$$t_{delay} = t_{trans} + t_{prop} + 8t_{slot} + 4t_{slot}. \tag{4.3}$$

This additional delay is incurred because the gateway allocates cluster heads with timeslots within the inter-cluster period based on the number of nodes within each respective cluster [28]. Since the first cluster contains four nodes, the added delay for the second cluster is $4t_{slot}$, and the expected delay for the nodes within the second cluster is 12.07 ms. Here, again, we observed an increase in delay as the complexity of the network increased. We observed an 8.234 ms delay for the first cluster and a 12.335 ms delay for the second. The results confirm the expected increase of the delay in the second cluster, but is higher than expected. This result is again believed to be due to resource constraints within MATLAB.

WIA-PA, WirelessHart, and ISA100.11A utilize TDMA for data communication within allocated timeslots [9]. While WirelessHart and ISA100.11A mainly implement a mesh-network configuration and our model of WIA-PA implements a mesh-star topology, the analysis of the delay in each network reduces to analyzing the delay within individual

timeslots over the number of hops between the node and the gateway. Since WirelessHart and ISA100.11A utilize graph-based routing, the delay equation becomes

$$t_{delay} = t_{trans} + t_{prop} + t_{slot} \tag{4.4}$$

for the single node case [9]. As the number of routing devices and nodes increase in a WirelessHart and ISA100.11A network, the delay also grows, but is determined by the routing algorithm and the order in which the devices are scheduled to transmit.

Within our model, the results from the end-to-end delay of the single cluster with a single node shows that there is a $7t_{slot}$ greater delay in WIA-PA. As the complexity of the network topology grows, the delay is deterministic based on the number of nodes in the network. However, the delay in WirelessHart-enabled and ISA100.11A-enabled networks is dependent upon the routing algorithm and when nodes are scheduled to transmit [9].

## 4.4 Received Packet Error Rate

Another key aspect of WIA-PA performance is the received packet error rate. To measure this rate, we count the number of data packets transmitted by a node and the number of packets correctly received by the gateway over a 24-hour period. We then calculate the percentage received by the gateway in error. We test the same three scenarios as in Sections 4.3.1-4.3.3.

### 4.4.1 Single Cluster Containing Single Node

During this scenario, a single node transmitted a packet every second, which led to 86,400 data frames transmitted during the 24-hour period. The gateway correctly received all 86,400 data frames, resulting in a 0% received packet error rate for this simulation.

### 4.4.2 Single Cluster Containing Eight Nodes

This simulation consisted of 691,200 data frames transmitted equally across the eight nodes within the same cluster. This time the gateway only received and correctly parsed 669,376 data frames. This simulation yielded an error rate of 3.16% for our model. This scenario simulates only one cluster, and each node within the cluster is allocated its own timeslot. We

expected to see the same results as in the simulation with only a single cluster and a single node. The increase in error from this simulation is believed to be due to MATLAB. As with the end-to-end delay results presented in Sections 4.3.1-4.3.3, the increase in number of nodes of the cluster also increases resource requirements in MATLAB and leads to increased error rates.

### 4.4.3 Two Clusters Containing Eight Nodes Total

This simulation consisted of 691,200 data frames transmitted equally across the eight nodes, but this time the nodes are separated into two different clusters. During this simulation, the gateway correctly parsed only 620,237 data frames. This resulted in an error rate of 10.27% for our model. Aside from the increase in theoretical delay, this simulation should also reduce to the single cluster with a single node case. The increase in incorrectly received packets is due to resource constraints within MATLAB as the complexity of the network increases.

## 4.5 Link Utilization

IEEE 802.15.4 allows for data to be transmitted utilizing either CSMA (during the CAP) or TDMA (within a GTS) [7]. Devices, based solely on IEEE 802.15.4, must request a GTS when a deterministic delay is desired. Therefore, there are two types of link utilization: contention based and link. WIA-PA, however, uses the CAP only for network management and TDMA for all data communication [28]. Similarly, WirelessHart and ISA100.11A use TDMA for data transmission for deterministic delay [9]. Link utilization within a WIA-PA enabled network, and an IEEE 802.15.4 GTS, is essentially a measure of timeslot utilization. This is due to the fact that WIA-PA nodes only transmit data during allocated timeslots [28]. To ensure maximum timeslot link utilization, the timeslot duration must equal the propagation duration of the data frame, when acknowledgement frames are not required. The timeslot duration $t_{ts}$ is defined by

$$t_{ts} = \frac{8maxPPDU}{R_b}, \tag{4.5}$$

where $maxPPDU$ is 133 bytes (as described in Section 3.1.3), and $R_b$ is the bit rate of 250 kbps [7].

Using

$$U = \left( \frac{8PPDU}{R_b t_{ts}} \right) \cdot 100, \tag{4.6}$$

we see that timeslot utilization $U$ is 100% when $t_{ts}$ is set to 4.256 ms and a frame with maximum data length is transmitted.

When acknowledgement frames are required, (4.5) is modified to

$$t_{ts} = \frac{8(maxPPDU + PPDU_{ack})}{R_b}, \tag{4.7}$$

where $PPDU_{ack}$ is 11 bytes long [7]. Using (4.6) and (4.7), we find that 4.608 ms is the minimum $t_{ts}$ timeslot duration required to maximize the timeslot utilization when including acknowledgement frames.

In our model, the $t_{ts}$ is 10 ms and the PPDU is 22 bytes long. Using (4.6), we find that timeslot utilization of our model is 7.04%. If we utilized a frame with maximum data length, our timeslot utilization increases to 42.56% [7]. If we require acknowledgement frames as well, we see that our timeslot utilization increases to 46.08%.

While the model's timeslot utilization is low, the battery life of the node is extended. This is because the node is in a sleep state when not transmitting during a timeslot [28]. The lower the $t_{ts}$, the shorter the superframe duration, and therefore the more often the node is active and not sleeping. The power efficiency can be improved by increasing the $t_{ts}$, thereby increasing the amount of time the device is in a sleep state [29]. Also, the timeslot utilization determined in our model is the same as the timeslot utilization found in a WirelessHart or an ISA100.11A network since they also use the IEEE 802.15.4 data link layer standard [9]. WirelessHart requires a 10 ms timeslot, thus the timeslot utilization would be exactly the same as that of WIA-PA [9]. The timeslot duration within an ISA100.11A network is configurable, but the utilization would also be the same provided a timeslot duration of 10 ms is used [9].

## 4.6 Chapter Summary

In this chapter, we compared the end-to-end delay of our model with the theoretical delay based on the WIA-PA standard. Our results indicate that the end-to-end delay of our model is consistent with the theoretical values in the simplest case. Our end-to-end delay results are also similar to the expected delays in WirelessHart and ISA100.11A networks [9]. However, as the number of nodes in our model is increased, the delay also increases. This increase is not due to the WIA-PA standard, but instead due to the computational expense in MATLAB. We also measured the received packet error rate for our model during a 24-hour period for three different situations. This error rate increased as we increased the number of nodes and complexity of the model. As with the measured delay, we believe that the rate of increase of the received packet error rate, between the different scenarios, is influenced by the computational expense of the model's complexity within MATLAB. Finally, we calculated the minimum timeslot duration to maximize the timeslot utilization. We determined that the timeslot utilization for our model is less than 50%, which leads to an increase in sleep duration of the nodes when compared to a higher timeslot utilization. This timeslot utilization is the same as expected in a WirelessHart network, as well as in an ISA100.11A network configured to use a timeslot with a duration of 10 ms [7].

THIS PAGE INTENTIONALLY LEFT BLANK

# CHAPTER 5:
# Conclusions and Future Work Recommendations

## 5.1 Summary and Conclusions

An ICS is a vital asset to the DOD. The addition of WSNs to an ICS has facilitated greater automation and network scalability. WIA-PA, WirelessHart, and ISA100.11A are all wireless standards that can be used in an ICS. However, the advantages of WIA-PA over WirelessHart and ISA100.11A make this standard a viable option for expanded use outside of China. To establish a performance base for this standard, we built a custom model in MATLAB. We have shown that our model provides end-to-end delay results comparable to the theoretical delay based on the WIA-PA standard, and confirms the deterministic capacity for data transmission of the WIA-PA standard. We also measured the error rate of our model. For the simple case, we observed no error between the number of data packets transmitted by the node and the number received by the gateway. As the number of nodes and complexity of the model was increased, however, we observed an increase in the error rate. Finally, we calculated the timeslot utilization of our model. We determined that our model maintained a very low timeslot utilization.

We expect WIA-PA to perform as well as WirelessHart and ISA100.11A with the added advantage of being able to coexist with other beacon-enabled networks. In the simplest scenario of our model, the measured results should be the same as expected in the other network implementations. The discrepancies we found in our simulations are believed to be due to the simulation environment, and not because of the WIA-PA implementation.

## 5.2 Future Work

In this thesis, we demonstrated that WIA-PA performs effectively in multiple network scenarios. However, there are areas of follow on research that would facilitate further analysis of the standard.

### 5.2.1 Network Simulator

MATLAB is not suited to efficiently simulate networks with network traffic. The implementation of WIA-PA using a network simulator, such as Contiki or the NS family, would make it more robust and provide more opportunities for in-depth analysis.

### 5.2.2 Validation of Security

The security of WIA-PA has not been characterized and the standard has not been subjected to various forms of attacks. A thorough security analysis needs to be undertaken to assess the standard's vulnerability and exploitability.

### 5.2.3 Routing

WIA-PA currently only supports static routing. By extending the standard to incorporate a dynamic routing protocol, such as Routing Protocol for Low-Power Lossy Networks (RPL), WIA-PA can be made more robust and instill self-healing capabilities.

# APPENDIX A: CSMA/CA

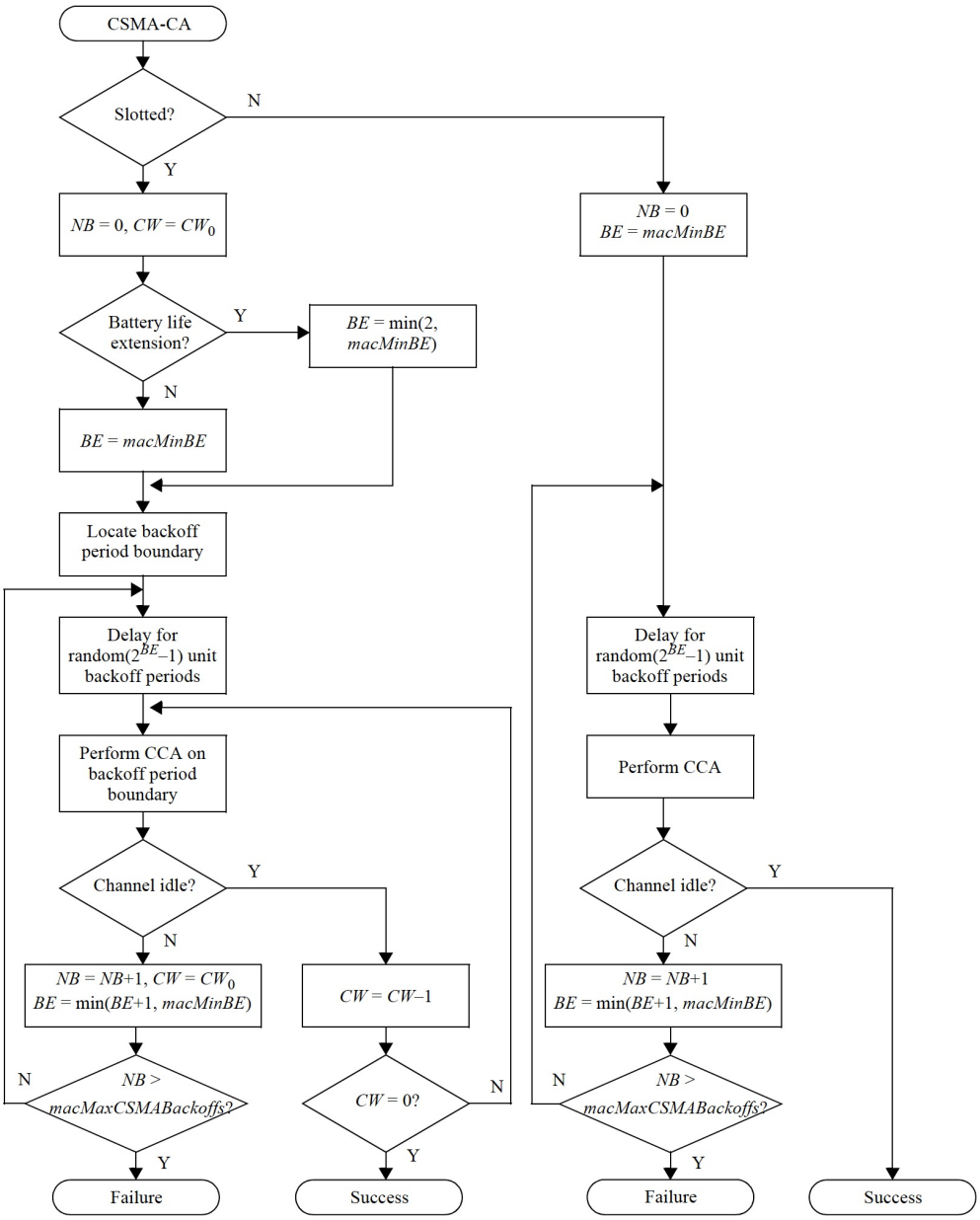


Figure A.1. CSMA/CA Algorithm. Source: [7].

THIS PAGE INTENTIONALLY LEFT BLANK

# APPENDIX B:
# MATLAB Code

## B.1 Classes

### B.1.1 wia_ffd.m

```
% wia_ffd.m in the @wia_ffd folder
% This is the class definition for Full Funciton Device (FFD)

classdef wia_ffd < handle
   properties
     name;
     my_ci;
     wia_cmd;
     pan_coord = 0;
     pan_coord_add = 0;
     parent_add = 0;
     cluster_id = '00';
     pan_id;
     assoc = 0;
     my_long_address;
     my_short_address = 'ffff';
     next_rt_address = '01';
     next_leaf_address = {{0}};
     rx = struct( 'fcf', [] , 'sf', [], 'p', [], 'ci', [], '...
        payload', []);
     tx = struct( 'fcf', [] , 'sf', [], 'p', [], 'ci', [], '...
        payload', []);
     bcn_payload;
     tx_frame=[];
     rx_frame=[];
     rec_frame=[];
     state=1;
     lock = 0;
     bcn_frame=[];
```

```
28      last_seq_num = 0;
29      d_seq_num = 0;
30      b_seq_num = 0;
31      ack_table = [];
32      beacon_t;
33      cc_t;
34      send_t;
35      asn_t;
36      parse_t;
37      p_t;
38      assoc_t;
39      rsn_t;
40      bcn_timestamp;
41      payload = strrep(num2str(ones(1,40)),' ','');
42      chan='';
43      n_table={};
44      waiting = 0;
45      join_table={{}};
46      send_buf = {};
47      send_data_buf = {};
48      direct = '';
49      buffer = {};
50      i;
51      pend_add = {};
52      asn_start=0;
53      rsn=0;
54      alloc;
55
56      count = 0;
57
58  end
59  methods
60      function set_name(obj,num)
61          obj.name = strcat('ffd_',num2str(num));
62      end
63
64      function set_pan_coord(obj)
65         obj.pan_coord = 1;
66         obj.my_short_address = '0000';
67         obj.pan_coord_add = obj.my_short_address;
```

```
68            obj.assoc = 1;
69            obj.asn_start = 0;
70        end
71
72        function set_long_add(obj, add)
73            obj.my_long_address =  strcat('ffffffff', dec2hex(add,8)...
                  );
74        end
75
76        function set_pan_id(obj, id)
77            if obj.pan_coord
78                obj.pan_id =  id;
79            else
80                disp('not pan_coord')
81                return
82            end
83        end
84
85        function struct_init(obj)
86            load('wia_structs.mat');
87            obj.rx=struct('fcf', fcf, 'sf', superframe, 'p', p, 'ci'...
                  , ci, 'payload', []);
88            obj.tx=struct('fcf', fcf, 'sf', superframe, 'p', p, 'ci'...
                  , ci, 'payload', []);
89            obj.bcn_payload = bp;
90            obj.my_ci = ci;
91            obj.wia_cmd = wia_cmd;
92            clear fcf sf p ci bp wia_cmd
93        end
94
95        function set_val(obj)
96            obj.my_ci.dev_type = 0;
97            obj.my_ci.power_src = 1;
98            obj.my_ci.rec_idle = 0;
99            obj.my_ci.sec_cap = 0;
100           obj.my_ci.allocate_add = 1;
101
102           obj.tx.sf.BO = 3;
103           obj.tx.sf.SO = obj.tx.sf.BO;
104           obj.tx.sf.finCapslot = 15;
```

```
105             obj.tx.sf.pan_coord = 1;
106
107             obj.tx.fcf.ack_req = 1;
108             obj.tx.fcf.dest_mode = 2;
109             obj.tx.fcf.src_mode = 2;
110
111             obj.tx.p.src_pan_id = obj.pan_id;
112             obj.tx.p.dest_pan_id = obj.pan_id;
113             obj.tx.p.src_add = obj.my_short_address;
114         end
115
116         function initialize(obj)
117             global ts
118             struct_init(obj);
119             set_val(obj);
120             delay = 0;
121             name = strcat(obj.name, '_beacon');
122             obj.beacon_t = timer('Name',name,'TimerFcn',@(¬,¬)beacon...
                    (obj),'StartDelay',delay,'ExecutionMode','fixedRate',...
                    'Period',32*ts);
123             name = strcat(obj.name, '_asn');
124             obj.asn_t = timer('Name',name,'TimerFcn',@(¬,¬)inc_asn(...
                    obj),'BusyMode', 'queue','ExecutionMode','fixedRate',...
                     'Period',ts);
125             name = strcat(obj.name, '_send');
126             obj.send_t = timer('Name',name,'TimerFcn',@(¬,¬)send(obj...
                    ),'BusyMode', 'drop','ExecutionMode','fixedRate','...
                    Period',ts/4);
127             name = strcat(obj.name, '_parse');
128             obj.parse_t = timer('Name',name,'TimerFcn',@(¬,¬)parse(...
                    obj),'ExecutionMode','fixedRate','BusyMode','drop','...
                    Period',ts/10);
129             name = strcat(obj.name, '_p');
130             obj.p_t = timer('Name',name,'TimerFcn',@(¬,¬)parse_wia(...
                    obj),'ExecutionMode','singleshot');
131             name = strcat(obj.name, '_assoc');
132             obj.assoc_t = timer('Name',name,'TimerFcn',@(¬,¬)...
                    assoc_func(obj),'ExecutionMode','fixedRate', 'Period'...
                    , ts/5);
133             name = strcat(obj.name, '_rsn');
```

```
134             obj.rsn_t = timer('Name',name,'TimerFcn',@(¬,¬)inc_rsn(...
                    obj),'ExecutionMode','fixedRate', 'StartDelay',1,'...
                    Period', ts);
135         end
136
137         function prop = get_prop(obj, property)
138             prop = obj.(property);
139         end
140
141         function set_prop(obj, property, value)
142             obj.(property) = value;
143         end
144
145        function start(obj)
146           if obj.pan_coord
147               asn_timer();
148               beacon_timer(obj);
149               parse_timer(obj);
150               send_timer(obj);
151           elseif ¬obj.pan_coord
152               assoc_timer(obj);
153               parse_timer(obj);
154               send_timer(obj)
155           end
156        end
157    end
158 end
```

### B.1.2 wia_rfd.m

```
1  % wia_rfd.m in the @wia_rfd folder
2  % This is the class definition for Reduced Funciton Device (RFD)
3
4  classdef wia_rfd < handle
5     properties
6       name;
7       my_ci;
8       wia_cmd;
```

```
9       pan_coord = 0;
10      pan_id;
11      pan_coord_add;
12      parent_add;
13      assoc = 0;
14      my_long_address;
15      my_short_address = 'ffff';
16      rx;
17      tx;
18      bcn_payload;
19      tx_frame = [];
20      rx_frame = [];
21      rec_frame=[''];
22      state=1;
23      lock = 0;
24      ack_table = [];
25      last_seq_num;
26      d_seq_num = 0;
27      cc_t;
28      rsn_t;
29      assoc_t;
30      parse_t;
31      send_t;
32      data_t
33      p_t;
34      bcn_timestamp = tic;
35      payload = strrep(num2str(ones(1,40)),' ','');
36      waiting = 0;
37      chan='';
38      n_table={{}};
39      direct = '';
40      buffer = {};
41      cluster_id;
42      asn_start = 0;
43      send_buf = {};
44      send_data_buf = {};
45      rsn = 0;
46      alloc;
47      count = 0;
48    end
```

```
49      methods
50          function set_name(obj,num)
51              obj.name = strcat('rfd_',num2str(num));
52          end
53
54          function struct_init(obj)
55              load('wia_structs.mat');
56              obj.rx = struct('fcf', fcf, 'sf', superframe, 'p', p, 'ci...
                    ', ci, 'payload', []);
57              obj.tx = struct('fcf', fcf, 'sf', superframe, 'p', p, 'ci...
                    ', ci, 'payload', []);
58              obj.wia_cmd = wia_cmd;
59              obj.bcn_payload = bp;
60              obj.my_ci = ci;
61              clear fcf sf p ci
62          end
63
64          function set_val(obj)
65              obj.my_ci.dev_type = 1;
66              obj.my_ci.power_src = 0;
67              obj.my_ci.rec_idle = 0;
68              obj.my_ci.sec_cap = 0;
69              obj.my_ci.allocate_add = 1;
70          end
71
72          function rsn = get_rsn(obj)
73              rsn = obj.rsn;
74          end
75
76          function initialize(obj)
77              global ts
78              struct_init(obj);
79              set_val(obj);
80              name = strcat(obj.name, '_send');
81              obj.send_t = timer('Name',name,'TimerFcn',@(¬,¬)send(obj)...
                    ,'BusyMode', 'drop','ExecutionMode','fixedRate','...
                    Period',ts/4);
82              name = strcat(obj.name, '_parse');
83              obj.parse_t = timer('Name',name,'TimerFcn',@(¬,¬)parse(...
                    obj),'ExecutionMode','fixedRate','BusyMode','drop','...
```

```
            Period',ts/4);
84          name = strcat(obj.name, '_p');
85          obj.p_t = timer('Name',name,'TimerFcn',@(¬,¬)parse_wia(...
                obj),'ExecutionMode','singleshot');
86          name = strcat(obj.name, '_data');
87          %obj.data_t = timer('Name',name,'TimerFcn',@(¬,¬)...
                send_data(obj),'ExecutionMode','fixedRate','StartDelay...
                ',15*ts,'BusyMode','drop','Period',32*ts);
88          obj.data_t = timer('Name',name,'TimerFcn',@(¬,¬)send_data...
                (obj),'ExecutionMode','fixedRate','BusyMode','drop','...
                Period',1);
89          name = strcat(obj.name, '_assoc');
90          obj.assoc_t = timer('Name',name,'TimerFcn',@(¬,¬)...
                assoc_func(obj),'ExecutionMode','fixedRate', 'Period',...
                 ts/10);
91          name = strcat(obj.name, '_rsn');
92          obj.rsn_t = timer('Name',name,'TimerFcn',@(¬,¬)inc_rsn(...
                obj),'ExecutionMode','fixedRate', 'StartDelay',1,'...
                TasksToExecute', 32, 'Period', ts);
93      end
94
95       function prop = get_prop(obj, property)
96           prop = obj.(property);
97       end
98
99       function set_prop(obj, property, value)
100          obj.(property) = value;
101      end
102
103     function set_long_add(obj, add)
104         obj.my_long_address =  strcat('ffffffff', dec2hex(add,8)...
                );
105      end
106
107     function start(obj)
108        assoc_timer(obj);
109        parse_timer(obj);
110        send_timer(obj);
111        data_timer(obj);
112     end
```

```
113     end
114 end
```

## B.2 Functions

### B.2.1 asn_timer.m

```
1  % asn_timer.m
2  % This function starts the time that controls the incrementing of ...
       the
3  % absolute slot number (ASN) for the network.  It can take in a ...
       value to
4  % initialize the ASN based on the ASN obtained from a beacon.
5
6  function asn_timer()
7      try
8          asn_t = timer('Name','asn','TimerFcn',@(¬,¬)inc_asn(),'...
               BusyMode', 'queue','StartDelay',1,'ExecutionMode','...
               fixedRate','Period',1);
9          start(asn_t);
10     catch e
11         fprintf('Caught exception: %s\n', e.message);
12     end
13 end
```

### B.2.2 assoc_func.m

```
1  % assoc_func.m
2  % This function is called by the assoc timer
3
4  function assoc_func(obj)
5      if obj.my_ci.dev_type == 0
6          if obj.pan_coord
7              return
8
9          elseif obj.assoc
```

```
10              stop(obj.assoc_t);
11              beacon_timer(obj);
12          else
13              obj.i = obj.i + 1;
14              pause(.05);  % a slight pause to let all the data ...
                    gather
15              if obj.i > 1000
16                  obj.set_pan_coord;
17                  obj.set_pan_id(hex2dec('dcba'));
18                  asn_timer(obj);
19                  obj.assoc = 1;
20              end
21          end
22
23      elseif obj.my_ci.dev_type == 1
24           if obj.assoc
25               stop(obj.assoc_t);
26               data_timer = timer('Name','client-data','TimerFcn',@(...
                     ¬,¬)send_data(obj),'ExecutionMode','fixedRate','...
                     Period',15);
27           end
28      end
29  end
```

### B.2.3 assoc_timer.m

```
1  % assoc_timer.m
2  % This timer checks the status of device association.
3
4  function assoc_timer(obj)
5      try
6          start(obj.assoc_t);
7      catch e
8          fprintf('Caught exception: %s\n', e.message);
9      end
10 end
```

### B.2.4 bcn_parse_wia.m

```
% bcn_parse_wia.m
% This function parses the payload of a beacon frame.

function bcn_parse_wia(obj)
    obj.bcn_payload.cluster_id = bin2dec(obj.rx.payload(1:8));
    obj.bcn_payload.asn = bin2dec(obj.rx.payload(9:56));
    obj.bcn_payload.time_val = bin2dec(obj.rx.payload(57:72));
    obj.bcn_payload.next_bcn_chan = bin2dec(obj.rx.payload(73:80))...
        ;
end
```

### B.2.5 beacon.m

```
% beacon.m
% This function is called by the beacon timer

function beacon(obj)
    obj.bcn_timestamp = tic;
    obj.tx.fcf.frame_type = 0;
    transmit(obj);
end
```

### B.2.6 beacon_timer.m

```
% beacon_timer.m
% This timer sets the frequency at which beacons are transmitted ...
    by a
% networking device.

function beacon_timer(obj)
    try
        start(obj.beacon_t);
    catch e
        fprintf('Caught exception: %s\n', e.message);
```

```
10      end
11  end
```

### B.2.7 cc_timer.m

```
1  % cc_timer.m
2  % This timer sets the frequency at which clear channel assessments...
       (CCA) are
3  % made.
4
5  function cc_timer(obj)
6      try
7          start(obj.cc_t);
8      catch e
9          fprintf('Caught exception: %s\n', e.message);
10     end
11 end
```

### B.2.8 channel.m

```
1  %channel.m
2  % This function simulates transmitted data through a channel. ...
       Currently it
3  % is noiseless. The signal is placed in neighbor channels after a ...
       delay to
4  % simulate transmit time.
5
6  function channel(obj,tx_sig)
7      delay = 100*length(tx_sig)/250000; % using increasing time by ...
           2 orders of magnitude for simulations
8      pause(delay)
9      for a=1:length(obj.n_table)
10         obj.n_table{a}.chan=tx_sig;
11         buffer = obj.n_table{a}.get_prop('buffer');
12         buffer{numel(buffer)+1} = tx_sig;
13         obj.n_table{a}.chan='';
```

```
14          obj.n_table{a}.set_prop('buffer', buffer);
15      end
16  end
```

### B.2.9 channel_check.m

```
1  % channel_check.m
2  % This function collects the data when the channel is no longer ...
       clear.
3
4  function channel_check(obj)
5      if ¬isempty(obj.chan)
6          if isempty(obj.buffer)
7              obj.buffer{1} = obj.chan;
8              obj.chan = '';
9          elseif ¬strcmp(obj.buffer{numel(obj.buffer)}, obj.chan)
10             obj.buffer{numel(obj.buffer)+1} = obj.chan;
11             obj.chan = '';
12         end
13     end
14 end
```

### B.2.10 check_assoc.m

```
1  % check_assoc.m
2  % This channel checks the association of the device and parses ...
       pertinent
3  % information.
4
5  function check_assoc(obj)
6      global t
7      cmd_parse_802154(obj);
8      if ¬strcmp(obj.my_short_address,'ffff')
9          obj.pan_id = obj.rx.p.src_pan_id;
10         obj.pan_coord_add = obj.rx.p.src_add;
11         obj.parent_add = obj.pan_coord_add;
```

```
        obj.tx.fcf.src_mode = 2;
        text = strcat(obj.name, '_associated');
        disp(text)
    end
end
```

### B.2.11 cmd_parse_802154.m

```
% cmd_parse_802154.m
% This function parses the payload of a IEEE 802.15.4 command ...
    frame.

function cmd_parse_802154(obj)
    switch obj.rx.p.cmd_type
        case 1
            obj.rx.ci.dev_type = bin2dec(obj.rx.payload(2));
            obj.rx.ci.power_src = bin2dec(obj.rx.payload(3));
            obj.rx.ci.rec_idle = bin2dec(obj.rx.payload(4));
            obj.rx.ci.sec_cap = bin2dec(obj.rx.payload(7));
            obj.rx.ci.allocate_add = bin2dec(obj.rx.payload(8));
        case 2
            % Association response 24 bits
            if obj.assoc
                obj.tx.payload = obj.rx.payload;
            else
                obj.my_short_address = hex_conv(obj.rx.payload...
                    (1:16),2);
                obj.cluster_id = hex_conv(obj.rx.payload(1:8),2);
                obj.assoc = bin2dec(obj.rx.payload(17:24))+1;
            end

        case 3
            % Disassociation reason 8 bits
            dis_reas = bin2dec(obj.rx.payload);

        case 5
            % Time Synchronization
            pan_id = obj.rx.payload(1:16);
```

```
29              coord_add = obj.rx.payload(17:32);
30              chan_num = obj.rx.payload(33:40);
31              short_add = obj.rx.payload(41:56);
32              chan_page = obj.rx.payload(57:64);
33
34          otherwise
35
36      end
37  end
```

### B.2.12 create_wia_frame.m

```
1  % create_wia_frame.m
2  % This function creates the MAC layer frame for IEEE 802.15.4.
3
4  function frame = create_wia_frame(obj)
5      global t
6
7      switch obj.tx.fcf.frame_type
8          case 0
9              frame = beacon_frame(obj);
10
11         case 1
12             frame = data_frame(obj, obj.tx.payload);
13
14         case 2
15             frame = ack_frame(obj);
16
17         case 3
18             frame = cmd_frame(obj);
19
20         otherwise
21             disp('error')
22     end
23 end
24
25 %% Beacon Frame
26 function frame = beacon_frame(obj)
```

```
27      global asn
28      if isempty(obj.send_buf)
29          obj.tx.fcf.frame_pend = 0;
30          obj.tx.p.pend_add = dec2bin(0,8);
31      else
32
33          obj.tx.fcf.frame_pend = 1;
34          temp_short = '';
35          temp_long = '';
36          i = 0;
37          j = 0;
38          for a = 1:numel(obj.pend_add)
39              if length(obj.pend_add{a}) == 4
40                  i = i+1;
41                  temp_short = strcat(temp_short,hex_conv(...
                        obj.pend_add{a},1));
42              elseif length(obj.pend_add{a}) == 16
43                  j = j+1;
44                  temp_long = strcat(temp_long,hex_conv(obj.pend_add...
                        {a},1));
45              end
46          end
47          obj.tx.p.pend_add = strcat(dec2bin(i,3), '0', dec2bin(j,3)...
                , '0', temp_short, temp_long);
48
49      end
50
51      fcf = strcat(dec2bin(obj.tx.fcf.frame_type,3), dec2bin(0,1), ...
            dec2bin(obj.tx.fcf.frame_pend,1) ...
52                      , dec2bin(0,1), dec2bin(1,1), dec2bin(0,3) ...
                            ...
53                      , dec2bin(0,2), dec2bin(obj.tx.fcf.version,2)...
                            , dec2bin(obj.tx.fcf.src_mode,2));
54
55      src_pan_id = dec2bin(obj.pan_id,16);
56      src_add = hex_conv(obj.my_short_address,1);
57      add_info = strcat(src_pan_id, src_add);
58
59      superframe_spec = strcat(dec2bin(obj.tx.sf.BO,4), dec2bin(...
            obj.tx.sf.SO,4), dec2bin(obj.tx.sf.finCapslot,4) ...
```

```
60                              , dec2bin(0,1), dec2bin(0,1), dec2bin(...
                                    obj.pan_coord,1) ...
61                              , dec2bin(obj.tx.sf.assoc_permit,1));
62
63      payload = strcat(dec2bin(obj.bcn_payload.cluster_id,8), ...
            dec2bin(asn,48), dec2bin(obj.bcn_payload.time_val,16), ...
            dec2bin(obj.bcn_payload.next_bcn_chan,8));
64
65      frame = strcat(fcf, dec2bin(obj.tx.p.b_seq_num,8), add_info, ...
            superframe_spec ...
66                , dec2bin(0,8), obj.tx.p.pend_add , payload);
67
68      obj.tx.p.b_seq_num = inc_seq_num(obj.tx.p.b_seq_num);
69
70  end
71
72  %% Data Frame
73  function frame = data_frame(obj,payload)
74      obj.tx.fcf.dest_mode = 2;
75      obj.tx.fcf.src_mode = 2;
76      obj.tx.p.src_add = obj.my_short_address;
77
78      dest_pan_id = dec2bin(obj.tx.p.dest_pan_id,16);
79      dest_add = hex_conv(obj.tx.p.dest_add,1);
80      src_add = hex_conv(obj.tx.p.src_add,1);
81      add_info = strcat(dest_pan_id, dest_add, src_add);
82
83      fcf = frame_ctrl(obj);
84
85      frame = strcat(fcf, dec2bin(obj.tx.p.d_seq_num,8), add_info, ...
            payload);
86
87      obj.tx.p.d_seq_num = inc_seq_num(obj.tx.p.d_seq_num);
88
89  end
90
91  %% Ack Frame
92  function frame = ack_frame(obj)
93
94      fcf = frame_ctrl(obj);
```

```
95
96      frame = strcat(fcf, dec2bin(obj.rx.p.d_seq_num,8));
97
98  end
99
100 %% Cmd Frame
101 function frame = cmd_frame(obj)
102 %      cmd = struct('assoc_req'      , 1 ...
103 %                   , 'assoc_res'      , 2 ...
104 %                   , 'disassoc'       , 3 ...
105 %                   , 'data_req'       , 4);
106
107     switch obj.tx.p.cmd_type
108         case 1 % association request
109             if obj.assoc
110                 dest_pan_id = dec2bin(obj.tx.p.dest_pan_id,16);
111                 if obj.pan_coord_add
112                     dest_add = hex_conv(obj.pan_coord_add,1);
113                 else
114                     dest_add = hex_conv(obj.parent_add,1);
115                 end
116                 src_add = hex_conv(obj.tx.p.src_add,1);
117                 add_info = strcat(dest_pan_id, dest_add, src_add);
118
119                 payload = strcat( dec2bin(0,1) , dec2bin(...
                        obj.tx.ci.dev_type,1) , dec2bin(...
                        obj.tx.ci.power_src,1) , dec2bin(...
                        obj.tx.ci.rec_idle,1) ...
120                      , dec2bin(0,2) , dec2bin(obj.tx.ci.sec_cap,1) ...
                           , dec2bin(obj.tx.ci.allocate_add,1) );
121             else
122                 obj.tx.fcf.ack_req = 1;
123                 obj.tx.fcf.dest_mode = 2;
124                 obj.tx.fcf.src_mode = 3;
125                 obj.tx.p.dest_pan_id = obj.pan_id;
126                 if obj.pan_coord_add
127                     obj.tx.p.dest_add = obj.pan_coord_add;
128                 else
129                     obj.tx.p.dest_add = obj.rx.p.src_add;
130                 end
```

```
131                 obj.tx.p.src_add = obj.my_long_address;
132                 dest_pan_id = dec2bin(obj.tx.p.dest_pan_id,16);
133                 dest_add = hex_conv(obj.tx.p.dest_add,1);
134                 src_add = hex_conv(obj.tx.p.src_add,1);
135                 add_info = strcat(dest_pan_id, dest_add, src_add);
136
137                 payload = strcat( dec2bin(0,1) , dec2bin(...
                        obj.tx.ci.dev_type,1) , dec2bin(...
                        obj.tx.ci.power_src,1) , dec2bin(...
                        obj.tx.ci.rec_idle,1) ...
138                      , dec2bin(0,2) , dec2bin(obj.tx.ci.sec_cap,1) ...
                           , dec2bin(obj.tx.ci.allocate_add,1) );
139             end
140
141         case 2 % association response
142             if length(obj.rx.p.src_add) == 16
143                 obj.tx.fcf.dest_mode = 3;
144             else
145                 obj.tx.fcf.dest_mode = 2;
146             end
147             dest_pan_id = dec2bin(obj.tx.p.dest_pan_id,16);
148             dest_add = hex_conv(obj.rx.p.src_add,1);
149             src_add = hex_conv(obj.my_short_address,1);
150             add_info = strcat(dest_pan_id, dest_add, src_add);
151             if obj.pan_coord
152                 address =  new_add(obj);
153                 payload = strcat( hex_conv(address,1) , dec2bin...
                        (0,8) );
154             else
155                 payload = obj.tx.payload;
156             end
157
158         case 3 % dissassociate
159
160         case 4 % data request
161             obj.tx.fcf.frame_pend = 0;
162             dest_pan_id = dec2bin(obj.tx.p.dest_pan_id,16);
163             dest_add = hex_conv(obj.rx.p.src_add,1);
164             if obj.assoc
165                 obj.tx.fcf.src_mode = 2;
```

```
166                 src_add = hex_conv(obj.my_short_address,1);
167             else
168                 obj.tx.fcf.src_mdoe = 3;
169                 src_add = hex_conv(obj.my_long_address,1);
170             end
171             add_info = strcat(dest_pan_id, dest_add, src_add);
172             payload = '';
173
174     end
175
176     fcf = frame_ctrl(obj);
177
178     frame = strcat(fcf, dec2bin(obj.tx.p.d_seq_num,8), add_info, ...
            dec2bin(obj.tx.p.cmd_type,8), payload);
179
180     obj.tx.p.d_seq_num = inc_seq_num(obj.tx.p.d_seq_num);
181
182 end
183
184 %% Frame Control Field
185 function fcf = frame_ctrl(obj)
186
187     fcf = strcat(dec2bin(obj.tx.fcf.frame_type,3), dec2bin(0,1), ...
            dec2bin(obj.tx.fcf.frame_pend,1) ...
188                     , dec2bin(obj.tx.fcf.ack_req,1), dec2bin(...
                            obj.tx.fcf.pan_comp,1), dec2bin(0,3) ...
189                     , dec2bin(obj.tx.fcf.dest_mode,2), dec2bin(...
                            obj.tx.fcf.version,2), dec2bin(...
                            obj.tx.fcf.src_mode,2));
190 end
```

### B.2.13 data_timer.m

```
1 % data_timer.m
2 % This function starts the time that controls the incrementing of ...
      the
3 % relative slot number (RSN) for the network.
4
```

```
function data_timer(obj)
    try
        start(obj.data_t);
    catch e
        fprintf('Caught exception: %s\n', e.message);
    end
end
```

### B.2.14 get_asn.m

```
% get_asn.m
% Function to retrieve the value of asn

function r = get_asn()
    global asn
    r = asn;
end
```

### B.2.15 hex_conv.m

```
% hex_conv.m
% This function converts bits to hex and hex to bits.  Makes data ...
    more
% readable. Currently required for storing and parsing of data.

function output = hex_conv(input, type)
    switch type
        case 1
            output = hex2bin(input);
        case 2
            output = bin2hex(input);
    end
end

function bin = hex2bin(input)
    bin = '';
```

```
    for a=1:length(input)
        bin = strcat(bin,dec2bin(hex2dec(input(a)),4));
    end
end

function hex = bin2hex(input)
    hex = '';
    for a=1:4:length(input)
       hex = strcat(hex, dec2hex(bin2dec(input(a:a+3)),1));
    end
    hex = lower(hex);
end
```

### B.2.16 inc_asn.m

```
% inc_asn.m
% This function increments the absolute slot number;

function inc_asn
    global asn
    global t_asn
    t_asn = tic;
    asn = asn+1;
    if ¬(asn ≤ hex2dec('fffffffffff'))
        asn = 0;
    end
end
```

### B.2.17 inc_seq_num.m

```
% inc_seq_num.m
% This function increments the sequence number of the MAC layer ...
    frames.

function new_seq_num = inc_seq_num(seq_num)
    new_seq_num = seq_num + 1;
```

```
    if ¬(new_seq_num ≤ 255)
        new_seq_num = 0;
    end
end
```

### B.2.18 new_add.m

```
% new_add.m
% This function produces 16 bit short addresses during the joining...
    process
% for requesting devices.

function next_add = new_add(obj)
    if length(obj.rx.p.src_add) == 16
        if obj.rx.ci.dev_type == 0 %ffd
            cluster_id = hex2dec(obj.next_rt_address);
            obj.next_leaf_address{cluster_id+1} = {0};
            next_add = strcat(dec2hex(cluster_id,2),dec2hex(...
                obj.next_leaf_address{cluster_id+1}{1},2));
            obj.next_rt_address = dec2hex(hex2dec(...
                obj.next_rt_address)+1);
        elseif obj.rx.ci.dev_type == 1 % rfd
            next_add = strcat(obj.cluster_id, dec2hex(numel(...
                obj.next_leaf_address{1}),2));
            obj.next_leaf_address{1}{hex2dec(next_add)+1} = numel(...
                obj.next_leaf_address{1});
        end
    elseif length(obj.rx.p.src_add) == 4
        cluster_id = obj.rx.p.src_add(1:2);
        ind_c = hex2dec(cluster_id);
        next_add = strcat(cluster_id, dec2hex(numel(...
            obj.next_leaf_address{ind_c+1}),2));
        ind_l = hex2dec(next_add(3:4));
        obj.next_leaf_address{ind_c+1}{ind_l+1} = numel(...
            obj.next_leaf_address{ind_c+1});
    end
end
```

### B.2.19 parse.m

```
% parse.m
% This function is called by the parse timer

function parse(obj)
    if ¬isempty(obj.buffer)
        obj.rec_frame = obj.buffer{1};
        obj.buffer(1) = [];
        parse_wia(obj);
    end
end
```

### B.2.20 parse_timer.m

```
% parse_timer.m
% This timer sets the frequency at which received data is parsed ...
    by the
% device.

function parse_timer(obj)
    try
        name = strcat(obj.name, '_parse');
        start(obj.parse_t)
    catch e
        fprintf('Caught exception: %s\n', e.message);
    end
end
```

### B.2.21 parse_wia.m

```
% parse_wia.m
% This function parses the IEEE 802.15.4 MAC layer. All higher ...
    level data
% will be in the payload when parsing is complete.

```

```
function parse_wia(obj)
    data = obj.rec_frame;
    obj.rec_frame = '';
    SHR = data(1:40);
    PHR = bin2dec(data(41:47));
    data = data(49:end);
    err = 0;

    if (err ~= 0)
        sprintf('FCS Failed!\n')
        return
    else
        last_bit = length(data);

        obj.rx.fcf.frame_type = bin2dec(data(1:3));
        obj.rx.fcf.sec = bin2dec(data(4)); % constant value
        obj.rx.fcf.frame_pend = bin2dec(data(5));
        obj.rx.fcf.ack_req = bin2dec(data(6));
        obj.rx.fcf.pan_comp = bin2dec(data(7)); % constant value
        obj.rx.fcf.dest_mode = bin2dec(data(11:12));
        obj.rx.fcf.version = bin2dec(data(13:14));
        obj.rx.fcf.src_mode = bin2dec(data(15:16));
        if obj.rx.fcf.frame_type == 0
            obj.rx.p.b_seq_num = bin2dec(data(17:24));
        else
            obj.rx.p.d_seq_num = bin2dec(data(17:24));
        end

        switch obj.rx.fcf.frame_type

            case 0 % beacon frame
                obj.rx.p.src_pan_id = bin2dec(data(25:40));
                if obj.rx.fcf.src_mode == 2 % 16 bit address
                    obj.rx.p.src_add = hex_conv(data(41:56),2);
                    obj.rx.sf.BO = bin2dec(data(57:60));
                    obj.rx.sf.SO = bin2dec(data(61:64));
                    obj.rx.sf.finCapslot = bin2dec(data(65:68));
                    obj.rx.sf.BLE = bin2dec(data(69)); % constant ...
                        value
                    obj.rx.sf.pan_coord = bin2dec(data(71));
```

```
44                      obj.rx.sf.assoc_permit = bin2dec(data(72));
45                      obj.rx.p.gts = bin2dec(data(73:80));
46                      obj.rx.p.pend_add = bin2dec(data(81:88));
47                      obj.rx.payload = data(89:last_bit);
48                  elseif obj.rx.fcf.src_mode == 3 % 64 bit address
49                      obj.rx.p.src_add = hex_conv(data(41:104),2);
50                      obj.rx.sf.BO = bin2dec(data(105:108));
51                      obj.rx.sf.SO = bin2dec(data(109:112));
52                      obj.rx.sf.finCapslot = bin2dec(data(113:116));
53                      obj.rx.sf.BLE = bin2dec(data(117)); % constant...
                            value
54                      obj.rx.sf.pan_coord = bin2dec(data(118));
55                      obj.rx.sf.assoc_permit = bin2dec(data(119));
56                      obj.rx.p.gts = data(120:128);
57                      obj.rx.p.pend_add = data(129:136);
58                      obj.rx.payload = data(136:last_bit);
59                      obj.rx.payload
60                  end
61
62              case 1 % data frame
63                  if obj.rx.fcf.dest_mode == 2 % 16 bit dest address
64                      obj.rx.p.dest_pan_id = bin2dec(data(25:40));
65                      obj.rx.p.dest_add = hex_conv(data(41:56),2);
66                      if obj.rx.fcf.src_mode == 2% 16 bit src ...
                            address
67                          obj.rx.p.src_add = hex_conv(data(57:72),2)...
                                ;
68                          obj.rx.payload = data(73:last_bit);
69                      elseif obj.rx.fcf.src_mode == 3 % 64 bit src ...
                            address
70                          obj.rx.p.src_add = hex_conv(data(57:120)...
                                ,2);
71                          obj.rx.payload = data(121:last_bit);
72                      elseif obj.rx.fcf.src_mode == 0 % no src ...
                            address
73                          obj.rx.payload = data(57:last_bit);
74                      end
75                  elseif obj.rx.fcf.dest_mode == 3 % 64 bit dest ...
                        address
76                      obj.rx.p.dest_pan_id = bin2dec(data(25:40));
```

```
77                      obj.rx.p.dest_add = hex_conv(data(41:104),2);
78                      if obj.rx.fcf.src_mode == 2 % 16 bit src ...
                            address
79                          obj.rx.p.src_add = hex_conv(data(105:120)...
                                ,2);
80                          obj.rx.payload = data(121:last_bit);
81                      elseif obj.rx.fcf.src_mode == 3 % 64 bit src ...
                            address
82                          obj.rx.p.src_add = hex_conv(data(105:168)...
                                ,2);
83                          obj.rx.payload = data(169:last_bit);
84                      elseif obj.rx.fcf.src_mode == 0 % no src ...
                            address
85                          obj.rx.payload = data(105:last_bit);
86                      end
87                  elseif obj.rx.fcf.dest_mode == 0 % no dest address
88                      if obj.rx.fcf.src_mode == 2 % 16 bit src ...
                            address
89                          obj.rx.p.src_pan_id = bin2dec(data(25:40))...
                                ;
90                          obj.rx.p.src_add = hex_conv(data(41:56),2)...
                                ;
91                          obj.rx.payload = data(57:last_bit);
92                      elseif obj.rx.fcf.src_mode == 3 % 64 bit src ...
                            address
93                          obj.rx.p.src_pan_id = bin2dec(data(25:40))...
                                ;
94                          obj.rx.p.src_add = hex_conv(data(41:104)...
                                ,2);
95                          obj.rx.payload = data(105:last_bit);
96                      elseif obj.rx.fcf.src_mode == 0 % no src ...
                            address
97                          obj.rx.payload = data(25:last_bit);
98                      end
99                  end
100
101             case 2 % ack frame
102
103             case 3 % cmd frame
104                 if obj.rx.fcf.dest_mode == 2 % 16 bit dest address
```

```
105                     obj.rx.p.dest_pan_id = bin2dec(data(25:40));
106                     obj.rx.p.dest_add = hex_conv(data(41:56),2);
107                     if obj.rx.fcf.src_mode == 2 % 16 bit src ...
                            address
108                         obj.rx.p.src_add = hex_conv(data(57:72),2)...
                                ;
109                         obj.rx.p.cmd_type = bin2dec(data(73:80));
110                         if ¬any([4,6,7] == obj.rx.p.cmd_type)
111                             obj.rx.payload = data(81:last_bit);
112                         end
113                     elseif obj.rx.fcf.src_mode == 3 % 64 bit src ...
                            address
114                         obj.rx.p.src_add = hex_conv(data(57:120)...
                                ,2);
115                         obj.rx.p.cmd_type = bin2dec(data(121:128))...
                                ;
116                         if ¬any([4,6,7] == obj.rx.p.cmd_type)
117                             obj.rx.payload = data(129:last_bit);
118                         end
119                     elseif obj.rx.fcf.src_mode == 0 % no src ...
                            address
120                         obj.rx.p.cmd_type = bin2dec(data(57:64));
121                         if ¬any([4,6,7] == obj.rx.p.cmd_type)
122                             obj.rx.payload = data(65:last_bit);
123                         end
124                     end
125                 elseif obj.rx.fcf.dest_mode == 3 % 64 bit dest ...
                        address
126                     obj.rx.p.dest_pan_id = bin2dec(data(25:40));
127                     obj.rx.p.dest_add = hex_conv(data(41:104),2);
128                     if obj.rx.fcf.src_mode == 2 % 16 bit src ...
                            address
129                         obj.rx.p.src_add = hex_conv(data(105:120)...
                                ,2);
130                         obj.rx.p.cmd_type = bin2dec(data(121:128))...
                                ;
131                         if ¬any([4,6,7] == obj.rx.p.cmd_type)
132                             obj.rx.payload = data(129:last_bit);
133                         end
```

```
134                     elseif obj.rx.fcf.src_mode == 3 %64 bit src ...
                            address
135                         obj.rx.p.src_add = hex_conv(data(105:168)...
                                ,2);
136                         obj.rx.p.cmd_type = bin2dec(data(169:176))...
                                ;
137                         if ¬any([4,6,7] == obj.rx.p.cmd_type)
138                             obj.rx.payload = data(177:last_bit);
139                         end
140                     elseif obj.rx.fcf.src_mode == 0 % no src ...
                            address
141                         obj.rx.p.cmd_type = bin2dec(data(105:112))...
                                ;
142                         if ¬any([4,6,7] == obj.rx.p.cmd_type)
143                             obj.rx.payload = data(113:last_bit);
144                         end
145                     end
146                 elseif obj.rx.fcf.dest_mode == 0 % no dest address
147                     if obj.rx.fcf.src_mode == 2 % 16 bit src ...
                            address
148                         obj.rx.p.src_pan_id = bin2dec(data(25:40))...
                                ;
149                         obj.rx.p.src_add = hex_conv(data(41:56),2)...
                                ;
150                         obj.rx.p.cmd_type = bin2dec(data(57:64));
151                         if ¬any([4,6,7] == obj.rx.p.cmd_type)
152                             obj.rx.payload = data(65:last_bit);
153                         end
154                     elseif obj.rx.fcf.src_mode == 3 % 64 bit src ...
                            address
155                         obj.rx.p.src_pan_id = bin2dec(data(25:40))...
                                ;
156                         obj.rx.p.src_add = hex_conv(data(41:104)...
                                ,2);
157                         obj.rx.p.cmd_type = bin2dec(data(105:112))...
                                ;
158                         if ¬any([4,6,7] == obj.rx.p.cmd_type)
159                             obj.rx.payload = data(113:last_bit);
160                         end
```

```
161                     elseif obj.rx.fcf.src_mode == 0 % no src ...
                            address
162                         obj.rx.p.cmd_type = bin2dec(data(25:32));
163                         if ¬any([4,6,7] == obj.rx.p.cmd_type)
164                             obj.rx.payload = data(33:last_bit);
165                         end
166                     end
167                 end
168 
169             otherwise
170                 disp('Illegal Frame_Type')
171                 return
172         end
173 
174         if any(strcmp(obj.rx.p.src_add, {obj.my_short_address, ...
                obj.my_long_address}))
175             return
176         elseif obj.rx.fcf.frame_type == 0 && ¬obj.pan_coord
177             parsed(obj)
178             return
179         elseif any(strcmp(obj.rx.p.dest_add, {obj.my_short_address...
                , obj.my_long_address, 'ffff'}))
180             parsed(obj)
181             return
182         end
183     end
184 end
```

### B.2.22 parsed.m

```
1 % parsed.m
2 % This function determines device responses and actions to ...
      received data.
3 
4 function parsed(obj)
5     global time_delay
6     if obj.pan_coord
```

```
7           if (obj.rx.fcf.ack_req == 1) && (¬any([0,2] == ...
                obj.rx.fcf.frame_type)) && ¬any(obj.rx.p.d_seq_num == ...
                obj.ack_table)
8               obj.last_seq_num = obj.rx.p.d_seq_num;
9               obj.tx.fcf.frame_type = 2;
10              transmit(obj);
11          end
12          if (obj.rx.fcf.frame_type == 1)
13              for a=1:numel(time_delay)
14                  if strcmp(obj.rx.payload, time_delay{a}{1})
15                      time_delay{a}{3} = toc(time_delay{a}{2});
16                      disp(time_delay{a}{3})
17                  end
18              end
19          end
20          if (obj.rx.fcf.frame_type == 3)
21             if (obj.rx.p.cmd_type == 1)
22                 cmd_parse_802154(obj);
23                 obj.tx.ci.dev_type = obj.rx.ci.dev_type;
24                 obj.tx.fcf.frame_type = 3;
25                 obj.tx.p.cmd_type = 2;
26                 tx = obj.tx;
27                 obj.pend_add{numel(obj.pend_add)+1} = ...
                       obj.rx.p.src_add;
28                 obj.send_buf{numel(obj.send_buf)+1} = {...
                       obj.rx.p.src_add,tx};
29             elseif (obj.rx.p.cmd_type == 4)
30                 if ¬isempty(obj.send_buf)
31                     if any(strcmp(obj.rx.p.src_add, obj.pend_add))
32                         for a=1:numel(obj.send_buf)
33                             if any(strcmp(obj.rx.p.src_add, ...
                                   obj.send_buf{a}{1}))
34                                 obj.tx = obj.send_buf{strcmp(...
                                       obj.rx.p.src_add, obj.send_buf{a...
                                       }{1})}{2};
35                                 obj.send_buf(strcmp(...
                                       obj.rx.p.src_add, obj.send_buf{a...
                                       }{1})) = [];
36                                 obj.rx.ci.dev_type = ...
                                       obj.tx.ci.dev_type;
```

```
37                                  obj.pend_add(strcmp(...
                                        obj.rx.p.src_add, obj.pend_add{a...
                                        })) = [];
38                                  transmit(obj);
39                                  break;
40                              end
41                          end
42                      else
43                          obj.tx.fcf.frame_type = 1;
44                          obj.tx.p.dest_add = obj.tx.p.src_add;
45                          obj.tx.payload = '';
46                          transmit(obj);
47                      end
48                  end
49              end
50          end
51      elseif ¬obj.pan_coord
52          if (obj.rx.fcf.ack_req == 1) && (¬any([0,2] == ...
                obj.rx.fcf.frame_type))
53              obj.tx.fcf.frame_type = 2;
54              transmit(obj);
55          end
56          if (obj.rx.fcf.frame_type == 0)
57              obj.bcn_timestamp = tic;
58              obj.rsn = 0;
59              obj.pan_id = obj.rx.p.src_pan_id;
60              if obj.rx.sf.pan_coord == 1
61                  obj.pan_coord_add = obj.rx.p.src_add;
62              end
63              if (obj.assoc == 0) && (obj.waiting == 0) && strcmp(...
                    obj.my_short_address, 'ffff')
64                  obj.tx.fcf.frame_type = 3;
65                  obj.tx.p.cmd_type = 1;
66                  obj.tx.ci = obj.my_ci;
67                  obj.waiting = 1;
68                  transmit(obj);
69              end
70              if obj.rx.fcf.frame_pend == 1
71                  if obj.rx.p.pend_add ≠ 0
72                      pend_add = dec2bin(obj.rx.p.pend_add,8);
```

```
73                      i = bin2dec(pend_add(1:3));
74                      j = bin2dec(pend_add(5:7));
75                      temp = {};
76                      if i ~= 0
77                          for a = 1:i
78                              temp{numel(temp)+1} = hex_conv(...
                                    obj.rx.payload(1:16),2);
79                              obj.rx.payload = obj.rx.payload(17:end...
                                    );
80                          end
81                      end
82                      if j ~= 0
83                          for a = 1:j
84                              temp{numel(temp)+1} = hex_conv(...
                                    obj.rx.payload(1:64),2);
85                              obj.rx.payload = obj.rx.payload(65:end...
                                    );
86                          end
87                      end
88                      if any(strcmp(obj.my_short_address, temp)) || ...
                        any(strcmp(obj.my_long_address, temp))
89                          obj.tx.fcf.frame_type = 3;
90                          obj.tx.p.cmd_type = 4;
91                          transmit(obj);
92                      end
93                  end
94              end
95              bcn_parse_wia(obj);
96          elseif (obj.rx.fcf.frame_type == 1) && obj.my_ci.dev_type ...
                == 0
97              %disp(obj.name)
98              obj.tx.fcf.frame_type = 1;
99              obj.tx.p.dest_add = obj.pan_coord_add;
100             obj.tx.fcf.ack_req = 0;
101             obj.tx.payload = obj.rx.payload;
102             transmit(obj);
103         elseif (obj.rx.fcf.frame_type == 3)
104             if obj.rx.p.cmd_type == 1 && obj.my_ci.dev_type == 0
105                 cmd_parse_802154(obj);
```

```
106                 obj.join_table{numel(obj.join_table)} = ...
                        obj.rx.p.src_add;
107                 obj.tx = obj.rx;
108                 obj.tx.p.src_add = obj.my_short_address;
109                 obj.tx.fcf.src_mode = 2;
110                 obj.waiting = 1;
111                 transmit(obj);
112             elseif obj.rx.p.cmd_type == 2
113                 if obj.assoc && obj.my_ci.dev_type == 0
114                     cmd_parse_802154(obj);
115                     obj.rx.p.src_add = obj.join_table{1};
116                     obj.join_table{1} = [];
117                     obj.tx = obj.rx;
118                     obj.tx.fcf.dest_mode = 3;
119                     obj.tx.p.src_add = obj.my_short_address;
120                     tx = obj.tx;
121                     obj.waiting = 0;
122                     obj.pend_add{numel(obj.pend_add)+1} = ...
                            obj.rx.p.src_add;
123                     obj.send_buf{numel(obj.send_buf)+1} = {...
                            obj.rx.p.src_add,tx};
124                 else
125                     check_assoc(obj);
126                 end
127             elseif (obj.rx.p.cmd_type == 4) && obj.my_ci.dev_type ...
                    == 0
128                 if ¬isempty(obj.send_buf)
129                     for a=1:numel(obj.send_buf)
130                         if (strcmp(obj.rx.p.src_add, obj.send_buf{a...
                                }{1}))
131                             obj.tx = obj.send_buf{strcmp(...
                                    obj.rx.p.src_add, obj.send_buf{a...
                                    }{1})}{2};
132                             transmit(obj)
133                         end
134                     end
135                 end
136             end
137         end
138     end
```

```
139 end
```

### B.2.23 send_data.m

```
1  % send_data.m
2  % This function sends data frame with an arbitrary payload.
3
4  function send_data(obj)
5      global asn
6      if mod(asn,32) == obj.alloc(1)-1
7          obj.tx.fcf.frame_type = 1;
8          obj.tx.fcf.ack_req = 0;
9          obj.tx.p.dest_add = strcat(obj.my_short_address(1:2),'00')...
               ;
10         obj.tx.payload  = strcat(hex_conv(obj.my_short_address,1),...
               dec2bin(obj.tx.p.d_seq_num,8),hex_conv('deadbeef',1));
11
12         transmit(obj);
13     end
14 end
```

### B.2.24 send.m

```
1  % send.m
2  % This function sends data during the allocated timeslot.
3  % It is also used to measure the end-to-end-delay.
4  function send(obj)
5      global asn
6      global time_delay
7      global t_asn
8
9      if mod(asn,32) == 0
10             obj.set_prop('count',0);
11     end
12     if ¬isempty(obj.send_data_buf)
13         tx_sig = obj.send_data_buf{1};
```

```
14          if mod(asn,32) == obj.alloc(obj.count+1)
15              obj.count = obj.count + 1;
16              pause(1-toc(t_asn))
17              start = tic;
18              if obj.tx.fcf.frame_type == 1 && obj.my_ci.dev_type ==...
                    1
19                  send_data(obj);
20                  time_delay{numel(time_delay)+1} = {obj.tx.payload,...
                        start};
21              end
22              obj.send_data_buf(1)=[];
23              channel(obj, tx_sig);
24          end
25      end
26  end
```

### B.2.25 send_timer.m

```
1  % send_timer.m
2  % This function starts the time that controls the incrementing of ...
       the
3  % relative slot number (RSN) for the network.
4
5  function send_timer(obj)
6      try
7          start(obj.send_t);
8      catch e
9          fprintf('Caught exception: %s\n', e.message);
10     end
11 end
```

### B.2.26 set_rec_val.m

```
1  % set_rec_val.m
2  % This function sets network information on the device based on a ...
       received
```

```
% beacon.

function set_rec_val(obj)
    obj.tx.sf.BO = obj.rx.sf.BO;
    obj.tx.sf.SO = obj.rx.sf.SO;
    obj.tx.sf.finCapslot = obj.rx.sf.finCapslot;
    obj.tx.sf.assoc_permit = obj.rx.sf.assoc_permit;

    obj.pan_id = obj.rx.p.src_pan_id;
    obj.tx.p.dest_pan_id = obj.rx.p.src_pan_id;
    obj.tx.p.dest_add = obj.rx.p.src_add;

    if obj.assoc
        obj.tx.fcf.src_mode = obj.rx.fcf.src_mode;
    end
    if (obj.rx.sf.pan_coord == 1)
       obj.pan_coord_add = obj.rx.p.src_add;
    end
end
```

### B.2.27 slotted_csma.m

```
% slotted_csma.m
% This function performs slotted carrier sense multple access / ...
    collision
% avoidance (CSMA/CA).

function status = slotted_csma(obj,frame)

    %disp('entered csma')
    NB = 0;
    CW = 2;
    bit_rate = 250000; %max bit_rate
    maxMacCSMABackoffs = 4; %default value
    macMaxBE = 5;
    macMinBE = 3;

    t_backoff = .032;%(20*4)/bit_rate; %symbols
```

```
16      t_superframe_slot = 1; %(3*t_backoff)*2^obj.tx.sf.SO;
17      N_backoff = 3*2^obj.tx.sf.SO;
18
19      BE = macMinBE;
20
21      while(NB < maxMacCSMABackoffs)
22          t_elapsed = toc(obj.bcn_timestamp);
23          backoff_boundary = (ceil(t_elapsed/t_backoff));
24          next_boundary = (backoff_boundary+randi([0,2^BE-1]));
25          pause(next_boundary*t_backoff)
26          disp(obj.chan)
27          while isempty(obj.chan)
28              CW = CW - 1;
29              next_boundary = next_boundary + 1;
30              if CW == 0
31                  channel(obj,frame);
32                  status = 1;
33                  return
34              end
35          end
36          CW = 2;
37          NB = NB + 1;
38          BE = min(BE+1, macMaxBE);
39      end
40      sprintf('Failure!\n')
41      status = 0;
42      return
43  end
```

### B.2.28 transmit.m

```
1  % transmit.m
2  % This function transmits the data during the proper timeslot ...
       based on
3  % which timeframe of the superframe it is in.
4
5  function transmit(obj)
6      global time_delay
```

```
7       global asn
8       type = {'beacon', 'data' ...
9              , 'ack', 'cmd'};
10      name = strcat(obj.name,'_',type{obj.tx.fcf.frame_type+1});
11      disp(name) % display
12      frame = create_wia_frame(obj);
13      preamble = '00000000000000000000000000000000';
14      SFD = '11100101';
15      SHR = strcat(preamble, SFD);
16      if length(frame) < 128*8
17          PHR = strcat(dec2bin(ceil(length(frame)/8),7),'0');
18      else
19          disp('frame is too long')
20      end
21
22      tx_sig = strcat(SHR, PHR, frame);
23      obj.tx_frame = tx_sig;
24      if any([1,3] == obj.tx.fcf.frame_type)
25        obj.ack_table(numel(obj.ack_table)+1) = obj.tx.p.d_seq_num;
26      end
27      if any(obj.tx.fcf.frame_type == [0,2])
28          channel(obj,tx_sig)
29      elseif  any(obj.tx.fcf.frame_type == 3)
30          status = slotted_csma(obj, tx_sig);
31              if status
32                  for a=1:numel(obj.send_buf)
33                      obj.send_buf(strcmp(obj.rx.p.src_add, ...
                           obj.send_buf{a}{1})) = [];
34                  end
35              end
36      elseif (obj.tx.fcf.frame_type == 1)
37          send_data_buf = obj.get_prop('send_data_buf');
38          send_data_buf{numel(send_data_buf)+1} = tx_sig;
39          obj.set_prop('send_data_buf',send_data_buf);
40      end
41  end
```

### B.2.29 wia_structs.m

```
% wia_structs.m
% These are structs to hold data for the network and makes for ...
    simpler
% device initialization.

superframe = struct(   'BO'           , 0 ...
                     , 'SO'           , 0 ...
                     , 'finCapslot'   , 0 ...
                     , 'BLE'          , 0  ...
                     , 'pan_coord'    , 0  ...
                     , 'assoc_permit' , 1);

fcf = struct(  'frame_type'   , 0 ...
             , 'sec'          , 0 ...
             , 'frame_pend'   , 0 ...
             , 'ack_req'      , 0 ...
             , 'pan_comp'     , 1 ...
             , 'dest_mode'    , 0 ...
             , 'version'      , 3 ...
             , 'src_mode'     , 0);

 p = struct(   'pend_add'     , 0 ...
             , 'd_seq_num'    , 0 ...
             , 'b_seq_num'    , 0 ...
             , 'src_pan_id'   , 0 ...
             , 'dest_pan_id'  , 0 ...
             , 'dest_add'     , 0 ...
             , 'src_add'      , 0 ...
             , 'cmd_type'     , 0);

 ci = struct(  'dev_type'     , 0 ...
             , 'power_src'    , 0 ...
             , 'rec_idle'     , 0 ...
             , 'sec_cap'      , 0 ...
             , 'allocate_add' , 0);

 bp = struct ( 'asn'           , 0 ...
             , 'cluster_id'    , 0 ...
             , 'time_val'      , 0 ...
```

```
              , 'next_bcn_chan' , 0);

wia_cmd = struct ( 'dlsl_fc'      , 0 ...
                 , 'dlsl_cmd_id'  , 0 ...
                 , 'cal_time_val' , 0);

sf_states = struct( 'cap'   , 0 ...
                  , 'cfp'   , 0 ...
                  , 'intra' , 0 ...
                  , 'inter' , 0 ...
                  , 'sleep' , 0);
```

### B.2.30 setup.m

```
% setup.m
% This script is used to initialize and test all of the devices

%%% global variables - section required for nodes
% global t
% t = tic;
% global ts
% ts = 1;
%
% global time_delay
% global t_asn;
% time_delay = {};
%
% global asn
% asn = 0;

%%% Tests

% %%% Single Cluster with 1 Node
% a=wia_ffd;
% b=wia_ffd;
% c=wia_rfd;
%
% a.set_pan_coord;
```

```
% initialize(a);
% a.set_pan_id(hex2dec('abcd'));
% a.name = 'ffd_1';
% a.set_long_add(1);
%
% b.name = 'ffd_2';
% initialize(b);
% b.pan_id = hex2dec('abcd');
% b.tx.p.src_pan_id = hex2dec('abcd');
% b.tx.p.dest_pan_id = hex2dec('abcd');
% b.set_long_add(2);
% b.my_short_address = '0100';
% b.assoc = 1;
% b.alloc = [24:31];
%
% c.name = 'rfd_1';
% initialize(c);
% c.pan_id =hex2dec('abcd');
% c.tx.p.src_pan_id = hex2dec('abcd');
% c.tx.p.dest_pan_id = hex2dec('abcd');
% c.set_long_add(4);
% c.my_short_address = '0101';
% c.assoc = 1;
% c.alloc=[16];
%
% a.n_table={b};
% b.n_table={a,c};
% c.n_table={b};
%
% a.start;
%
% % cluster 1
% b.start;
% c.start;

%%% Single Cluster with 8 Nodes
% a=wia_ffd;
% b=wia_ffd;
% c=wia_rfd;
% d=wia_rfd;
```

```
% e=wia_rfd;
% f=wia_rfd;
% g=wia_rfd;
% h=wia_rfd;
% i=wia_rfd;
% j=wia_rfd;
%
% a.set_pan_coord;
% initialize(a);
% a.set_pan_id(hex2dec('abcd'));
% a.name = 'ffd_1';
% a.set_long_add(1);
% a.alloc = [0];
%
% b.name = 'ffd_2';
% initialize(b);
% b.pan_id = hex2dec('abcd');
% b.tx.p.src_pan_id = hex2dec('abcd');
% b.tx.p.dest_pan_id = hex2dec('abcd');
% b.set_long_add(2);
% b.my_short_address = '0100';
% b.assoc = 1;
% b.alloc = [24:31];
%
% c.name = 'rfd_1';
% initialize(c);
% c.pan_id = hex2dec('abcd');
% c.tx.p.src_pan_id = hex2dec('abcd');
% c.tx.p.dest_pan_id = hex2dec('abcd');
% c.set_long_add(3);
% c.my_short_address = '0101';
% c.assoc = 1;
% c.alloc = [16];
%
% d.name = 'rfd_2';
% initialize(d);
% d.pan_id =hex2dec('abcd');
% d.tx.p.src_pan_id = hex2dec('abcd');
% d.tx.p.dest_pan_id = hex2dec('abcd');
% d.set_long_add(4);
```

```
105 % d.my_short_address = '0102';
106 % d.assoc = 1;
107 % d.alloc=[17];
108 % %
109 % e.name = 'rfd_3';
110 % initialize(e);
111 % e.pan_id =hex2dec('abcd');
112 % e.tx.p.src_pan_id = hex2dec('abcd');
113 % e.tx.p.dest_pan_id = hex2dec('abcd');
114 % e.set_long_add(5);
115 % e.my_short_address = '0103';
116 % e.assoc = 1;
117 % e.alloc = [18];
118 % %
119 % f.name = 'rfd_4';
120 % initialize(f);
121 % f.pan_id =hex2dec('abcd');
122 % f.tx.p.src_pan_id = hex2dec('abcd');
123 % f.tx.p.dest_pan_id = hex2dec('abcd');
124 % f.set_long_add(6);
125 % f.my_short_address = '0104';
126 % f.assoc = 1;
127 % f.alloc = [19];
128 % %
129 % g.name = 'rfd_5';
130 % initialize(g);
131 % g.pan_id =hex2dec('abcd');
132 % g.tx.p.src_pan_id = hex2dec('abcd');
133 % g.tx.p.dest_pan_id = hex2dec('abcd');
134 % g.set_long_add(7);
135 % g.my_short_address = '0105';
136 % g.assoc = 1;
137 % g.alloc = [20];
138 % %
139 % h.name = 'rfd_6';
140 % initialize(h);
141 % h.pan_id =hex2dec('abcd');
142 % h.tx.p.src_pan_id = hex2dec('abcd');
143 % h.tx.p.dest_pan_id = hex2dec('abcd');
144 % h.set_long_add(8);
```

```
145 % h.my_short_address = '0106';
146 % h.assoc = 1;
147 % h.alloc = [21];
148 % %
149 % i.name = 'rfd_7';
150 % initialize(i);
151 % i.pan_id =hex2dec('abcd');
152 % i.tx.p.src_pan_id = hex2dec('abcd');
153 % i.tx.p.dest_pan_id = hex2dec('abcd');
154 % i.set_long_add(9);
155 % i.my_short_address = '0107';
156 % i.assoc = 1;
157 % i.alloc = [22];
158 % %
159 % j.name = 'rfd_8';
160 % initialize(j);
161 % j.pan_id =hex2dec('abcd');
162 % j.tx.p.src_pan_id = hex2dec('abcd');
163 % j.tx.p.dest_pan_id = hex2dec('abcd');
164 % j.set_long_add(10);
165 % j.my_short_address = '0108';
166 % j.assoc = 1;
167 % j.alloc = [23];
168 %
169 % a.n_table={b};
170 % b.n_table={a,c,d,e,f,g,h,i,j};
171 % c.n_table={b};
172 % d.n_table={b};
173 % e.n_table={b};
174 % f.n_table={b};
175 % g.n_table={b};
176 % h.n_table={b};
177 % i.n_table={b};
178 % j.n_table={b};
179 %
180 % a.start;
181 %
182 % % cluster 1
183 % b.start;
184 % c.start;
```

```
185 % d.start;
186 % e.start;
187 % f.start;
188 % g.start;
189 % h.start;
190 % i.start;
191 % j.start;
192 %
193 %%% Two Clusters with 4 nodes each
194 % a=wia_ffd;
195 % b=wia_ffd;
196 % c=wia_ffd;
197 %
198 % d=wia_rfd;
199 % e=wia_rfd;
200 % f=wia_rfd;
201 % g=wia_rfd;
202 % h=wia_rfd;
203 % i=wia_rfd;
204 % j=wia_rfd;
205 % k=wia_rfd;
206 %
207 % a.set_pan_coord;
208 % initialize(a);
209 % a.set_pan_id(hex2dec('abcd'));
210 % a.name = 'ffd_1';
211 % a.set_long_add(1);
212 %
213 % b.name = 'ffd_2';
214 % initialize(b);
215 % b.pan_id = hex2dec('abcd');
216 % b.pan_coord_add = '0000';
217 % b.tx.p.src_pan_id = hex2dec('abcd');
218 % b.tx.p.dest_pan_id = hex2dec('abcd');
219 % b.set_long_add(2);
220 % b.my_short_address = '0100';
221 % b.assoc = 1;
222 % b.alloc = [24:27];
223 %
224 % c.name = 'ffd_3';
```

```
% initialize(c);
% c.pan_id = hex2dec('abcd');
% c.pan_coord_add = '0000';
% c.tx.p.src_pan_id = hex2dec('abcd');
% c.tx.p.dest_pan_id = hex2dec('abcd');
% c.set_long_add(3);
% c.my_short_address = '0200';
% c.assoc = 1;
% c.alloc = [28:31];
%
% d.name = 'rfd_1';
% initialize(d);
% d.pan_id =hex2dec('abcd');
% d.tx.p.src_pan_id = hex2dec('abcd');
% d.tx.p.dest_pan_id = hex2dec('abcd');
% d.set_long_add(4);
% d.my_short_address = '0101';
% d.assoc = 1;
% d.alloc=[16];
% %
% e.name = 'rfd_2';
% initialize(e);
% e.pan_id =hex2dec('abcd');
% e.tx.p.src_pan_id = hex2dec('abcd');
% e.tx.p.dest_pan_id = hex2dec('abcd');
% e.set_long_add(5);
% e.my_short_address = '0102';
% e.assoc = 1;
% e.alloc = [17];
% %
% f.name = 'rfd_3';
% initialize(f);
% f.pan_id =hex2dec('abcd');
% f.tx.p.src_pan_id = hex2dec('abcd');
% f.tx.p.dest_pan_id = hex2dec('abcd');
% f.set_long_add(6);
% f.my_short_address = '0103';
% f.assoc = 1;
% f.alloc = [18];
% %
```

```
265 % g.name = 'rfd_4';
266 % initialize(g);
267 % g.pan_id =hex2dec('abcd');
268 % g.tx.p.src_pan_id = hex2dec('abcd');
269 % g.tx.p.dest_pan_id = hex2dec('abcd');
270 % g.set_long_add(7);
271 % g.my_short_address = '0104';
272 % g.assoc = 1;
273 % g.alloc = [19];
274 % %
275 % h.name = 'rfd_5';
276 % initialize(h);
277 % h.pan_id =hex2dec('abcd');
278 % h.tx.p.src_pan_id = hex2dec('abcd');
279 % h.tx.p.dest_pan_id = hex2dec('abcd');
280 % h.set_long_add(8);
281 % h.my_short_address = '0201';
282 % h.assoc = 1;
283 % h.alloc = [16];
284 % %
285 % i.name = 'rfd_6';
286 % initialize(i);
287 % i.pan_id =hex2dec('abcd');
288 % i.tx.p.src_pan_id = hex2dec('abcd');
289 % i.tx.p.dest_pan_id = hex2dec('abcd');
290 % i.set_long_add(9);
291 % i.my_short_address = '0202';
292 % i.assoc = 1;
293 % i.alloc = [17];
294 % %
295 % j.name = 'rfd_7';
296 % initialize(j);
297 % j.pan_id =hex2dec('abcd');
298 % j.tx.p.src_pan_id = hex2dec('abcd');
299 % j.tx.p.dest_pan_id = hex2dec('abcd');
300 % j.set_long_add(10);
301 % j.my_short_address = '0203';
302 % j.assoc = 1;
303 % j.alloc = [18];
304 % %
```

```
% k.name = 'rfd_8';
% initialize(k);
% k.pan_id =hex2dec('abcd');
% k.tx.p.src_pan_id = hex2dec('abcd');
% k.tx.p.dest_pan_id = hex2dec('abcd');
% k.set_long_add(11);
% k.my_short_address = '0204';
% k.assoc = 1;
% k.alloc = [19];
%
%
% a.n_table={b};
% b.n_table={a,d,e,f,g};
% c.n_table={a,h,i,j,k};
% d.n_table={b};
% e.n_table={b};
% f.n_table={b};
% g.n_table={b};
% h.n_table={c};
% i.n_table={c};
% j.n_table={c};
% k.n_table={c};
%
% a.start;

% cluster 1
% b.start;
% d.start;
% e.start;
% f.start;
% g.start;

% cluster 2
% c.start;
% h.start;
% i.start;
% j.start;
% k.start;
```

THIS PAGE INTENTIONALLY LEFT BLANK

# List of References


[1] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and H. Adam, "Sp 800-82 revision 2. Guide to industrial control systems (ics) security: Supervisory control and data acquisition (scada) systems, distributed control systems (dcs), and other control system configurations such as programmable logic controllers (plc)," Gaithersburg, MD, 2015.

[2] M. Chipley and D. Haegley. (n.d.). Cybersecuring Industrial Control Systems. The Military Engineer. [Online]. Available: http://themilitaryengineer.com/index.php/component/k2/item/261-cybersecuring-industrial-control-systems. Accessed Aug. 28, 2017.

[3] *Navy and Marine Corps Industrial Control Systems Monitoring Stations*, FC 4-141-05N, Department of Defense, Washington, DC, 2015, pp. 1–37.

[4] B. Galloway and G. P. Hancke, "Introduction to industrial control networks," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 860–880, 2013.

[5] A. Willig, K. Matheus, and A. Wolisz, "Wireless technology in industrial networks," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1130–1151, June 2005.

[6] A. Nechibvute and C. Mudzingwa, "Wireless sensor networks for scada and industrial control systems," *International Journal of Engineering and Technology*, vol. 3, no. 12, pp. 1025–1035, December 2013.

[7] "Ieee standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans)," *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pp. 1–314, Sept 2011.

[8] M. Nobre, I. Silva, and L. Guedes, "Routing and scheduling algorithms for wirelesshart networks: A survey," *Sensors*, no. 15, pp. 9703–9740, Apr 2015.

[9] Q. Wang and J. Jiang, "Comparative examination on architecture and protocol of industrial wireless sensor network standards," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2197–2219, 2016.

[10] M. Zheng, W. Liang, H. Yu, and Y. Xiao, "Performance analysis of the industrial wireless networks standard: Wia-pa," *Mobile Networks and Applications*, vol. 22, no. 1, pp. 139–150, Feb 2017.

[11] *Transmission of IPv6 Packets over WIA-PA Networks*, draft-wang-6lo-wiapa-04, 2016.

[12] P. Haibo Zhang, M. Soldati, and M. Johansson, "Performance bounds and latency-optimal scheduling for convergecast in wirelesshart networks," *IEEE Transactions on Wireless Communications*, vol. 12, no. 6, pp. 2688–2696, June 2013.

[13] Z. Z. Wu, "Research of wirelesshart network layer routing algorithm," *Applied Mechanics and Materials*, vol. 336-338, pp. 1827–1832, July 2013.

[14] J. Ersvik, M. Gidlund, A. Ahlén, and T. Nyberg, "Analysis of reliability and energy consumption in industrial wireless sensor networks," Uppsala universitet, Teknisk-naturvetenskapliga vetenskapsområdet, Tekniska sektionen, Institutionen för teknikvetenskaper, Signaler och System, 2012.

[15] Z. Sheng, W. Xuanzhao, and L. Dongdong, "An improved low power time synchronization algorithm for wirelesshart network," in *2013 UKSim 15th International Conference on Computer Modelling and Simulation (UKSim)*, 2013, pp. 672–676.

[16] M. Hua and L. Dong, "A closed-loop adjusting strategy for wirelesshart time synchronization," in *2011 11th International Symposium on Communications and Information Technologies (ISCIT)*, 2011, pp. 131–135.

[17] M. Fang, Y. Sun, X. Zhang, and J. Sun, "Elots: Energy-efficient local optimization time synchronization algorithm for wirelesshart networks," in *2014 8th Annual IEEE Systems Conference (SysCon)*, 2014, pp. 402–406.

[18] A. Depari, P. Ferrari, A. Flammini, and E. Sisinni, "Introducing a simulation tool for wirelesshart networks," *IFAC Proceedings Volumes*, vol. 42, no. 3, pp. 234–237, 2009.

[19] P. Zand, A. Dilo, and P. Havinga, "Implementation of wirelesshart in ns-2 simulator," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, Sept 2012, pp. 1–8.

[20] F. Rezha and F. Soo Young Shin, "Performance evaluation of isa100.11a industrial wireless network," in *IET Conference Proceedings*. Stevenage: The Institution of Engineering and Technology, 2013.

[21] Y. Serizawa, T. Yano, M. Miyazaki, K. Mizugaki, R. Fujiwara, and M. Kokubo, "Verification of interference avoidance effect with adaptive channel diversity method based on isa100.11a standard," in *2013 IEEE Radio and Wireless Symposium (RWS)*, 2013, pp. 361–363.

[22] M. Miyazaki, R. Fujiwara, K. Mizugaki, and M. Kokubo, "Adaptive channel diversity method based on isa100.11a standard for wireless industrial monitoring," in *2012 IEEE Radio and Wireless Symposium (RWS)*, 2012, pp. 131–134.

[23] T. Nhon and D.-S. Kim, "Traffic-aware message scheduling method for isa100.11a," in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, 2013, pp. 649–654.

[24] Y. Wei and D.-S. Kim, "A self-stabilized firefly synchronization method for the isa100.11a network," in *2013 International Conference on ICT Convergence (ICTC)*, 2013, pp. 881–886.

[25] Y. Zhou, Q. Wang, and Y. Wan, *High Accurate Time Synchronization Mechanism for WIA-PA Network*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 93–104.

[26] X. Jin and P. Zeng, "A fast real-time scheduling algorithm for wia-pa," *Applied Mechanics and Materials*, vol. 519-520, pp. 124–127, 2014.

[27] L. Meng and X. Du, "Research of industrial wireless network wia-pa multi-path routing protocol wmdsr," in *2011 International Symposium on Computer Science and Society (ISCCS)*, 2011, pp. 51–54.

[28] *Industrial networks – Wireless communication network and communication profiles – WIA-PA*, IEC 62601:2015, 2015.

[29] Y. Dongfeng and X. Yan-qun, "Design and implementation of low power consumption based on wia-pa networks," in *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, Aug 2010, vol. 4, pp. 572–575.

THIS PAGE INTENTIONALLY LEFT BLANK

# Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California